DESIGN AND IMPLEMENTATION OF A GRAPHICS PACKAGE IN SIMULA–PART II

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

By

K. SIVA KRISHNA REDDY

to the

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPUR

APRIL, 1985

CERTIFICATE

This is to certify that the thesis titled 'DESIGN AND IMPLEMENTATION OF A GRAPHICS PACKAGE IN SIMULA - PART II' has been carried out by Shri K. Siva Krishna Reddy under our supervision and that this has not been submitted elsewhere for an award of a degree.

R. Raghuram

(Dr. R. Raghuram)
Assistant Professor
Dept. of Electrical Engg.
Indian Institute of Technology
Kanpur

Dhande

(Dr. Sanjay G. Dhande)
Assistant Professor
Dept. of Computer Science and Engineering,
Indian Institute of Technology
Kanpur

ACKNOWLEDGEMENT

I take this opportunity to express my heartfelt thanks to

- my thesis supervisors Dr. R. Raghuram and Dr. S.G. Dhande for suggesting this topic and offering their valuable guidance throughout the thesis work.
- my director Dr. E. Bhagiratha Rao, and my senior officers Brig. B.Y. Sankara Narayana and Sri C.K. Sukumaran for providing me the opportunity to do the M.Tech. course.
- my colleague Mr. M.M. Ahmad for extending timely help whenever needed.
- my friends, especially Mr. L.R. Nagamoorthy for making my stay at IITK memorable.
- Mr. J.S. Rawat for his neat typing of this thesis.

K. SIVA KRISHNA REDDY

ABSTRACT

A graphics software package for TEKTRONIX 4006-1 graphics terminal has been designed in SIMULA language and implemented using DEC-1090 system as the host computer. It consists of two parts. The first part called 'GSIMULA' deals with displaying images of two dimensional objects. The second part called 'GRAS' deals with displaying images of three dimensional objects. This thesis is concerned with the second part. The facilities provided by the package GRAS are: (1) Different types of planar geometric projections (2) Instant transformations (3) Picture segmentation (4) Hidden line/surface elimination. These facilities are made available through a compact command set. The package automatically detects and corrects minor errors committed by the application programmer. The application programmer is merely warned. Execution is terminated when the error is severe. For debugging and checking the package, demonstration programs have been developed. These demonstration programs illustrate typical usage of most of the commands. Except for one procedure, the entire package is written in SIMULA, a high-level language and hence this package can easily be implemented on any machine that has a SIMULA compiler. With minor modifications, the package can be used with any other graphic output device.

CONTENTS

		Page
Chapter 1	INTRODUCTION	1
	1.1 Software of graphics system	1
	1.2 Simula and its features	5
	1.3 State of the art of the packages	6
	1.4 Objective and scope of present work	7
Chapter 2	SPECIFICATIONS	10
	2.1 Output functions	10
	2.2 Viewing functions	13
	2.3 Instance Transformation functions	20
	2.4 Picture segmentation functions	22
	2.5 Control and other functions	24
Chapter 3	IMPLEMENTATION DETAILS	25
	3.1 Instance transformation	25
	3.2 View plane transformation	27
	3.3 Clipping	30
	3.4 Parallel projection	34
	3.5 Perspective projection	35
	3.6 Hidden line elimination	37
	3.7 Windowing transformation	40

		Page
Chapter 4	CASE STUDIES	44
	4.1 Sphere	44
	4.2 Ring ball	45
	4.3 Tyre	45
	4.4 Housing complex	46
	4.5 Hollow cube	46
	4.6 Projections	47
Chapter 5	CONCLUSIONS	55
	5.1 Technical summary	55
	5.2 Further scope	56
Appendix A	3-d Transformations	57
Appendix B	Planar geometric projections	63
References		71
Program listing		
Demostration programs		

CHAPTER 1

INTRODUCTION

The aim of graphics system design is to simplify the writing of graphic application programs. A graphics system may be defined as any collection of hardware and software designed to make it easier to use graphic input and output in computer programs. The design of graphics systems is a very important aspect of computer graphics. Without such systems, graphics application programs would be extremely difficult to write; only the most expert programmers would be competent to write them, and their rate of software production would be very slow. It is only by constructing graphics systems that we make it possible to exploit the potential uses of computer graphics.

1.1 SOFTWARE OF GRAPHICS SYSTEM:

There are two approaches in the design of graphics system. One approach is the use of subroutines or procedures to access the capabilities of graphics system. The other approach is to extend the existing programming language or design a new language with special statements and programming constructs for graphical input and output. Our design is based on the first approach and the resulting

graphics system is called 'graphics package'. It consists of a set of subroutines or procedures which can be used by an application programmer to generate pictures on the screen of display devices and to handle graphical interaction. It shields the application programmer from needing to know the specific low level architecture of the display device and XY-coordinate system of physical screen. A programmer's model of how the package functions, is shown schematically in Fig. 1.1.

It consists of two major components: Hardware and software. The hardware component is the host computer and the graphics display terminal; the host's CPU and the memory shared by CPU and DPU are not shown here to reduce the complexity of the diagram. The software consists of three components.

1. Application Data Structure
2. Application Program
3. Graphics system (graphics package)

The application programmer first constructs an application model of the objects that the user will manipulate and view, and stores it in application data structure. The application model typically contains geometric coordinate data that define the shape of the

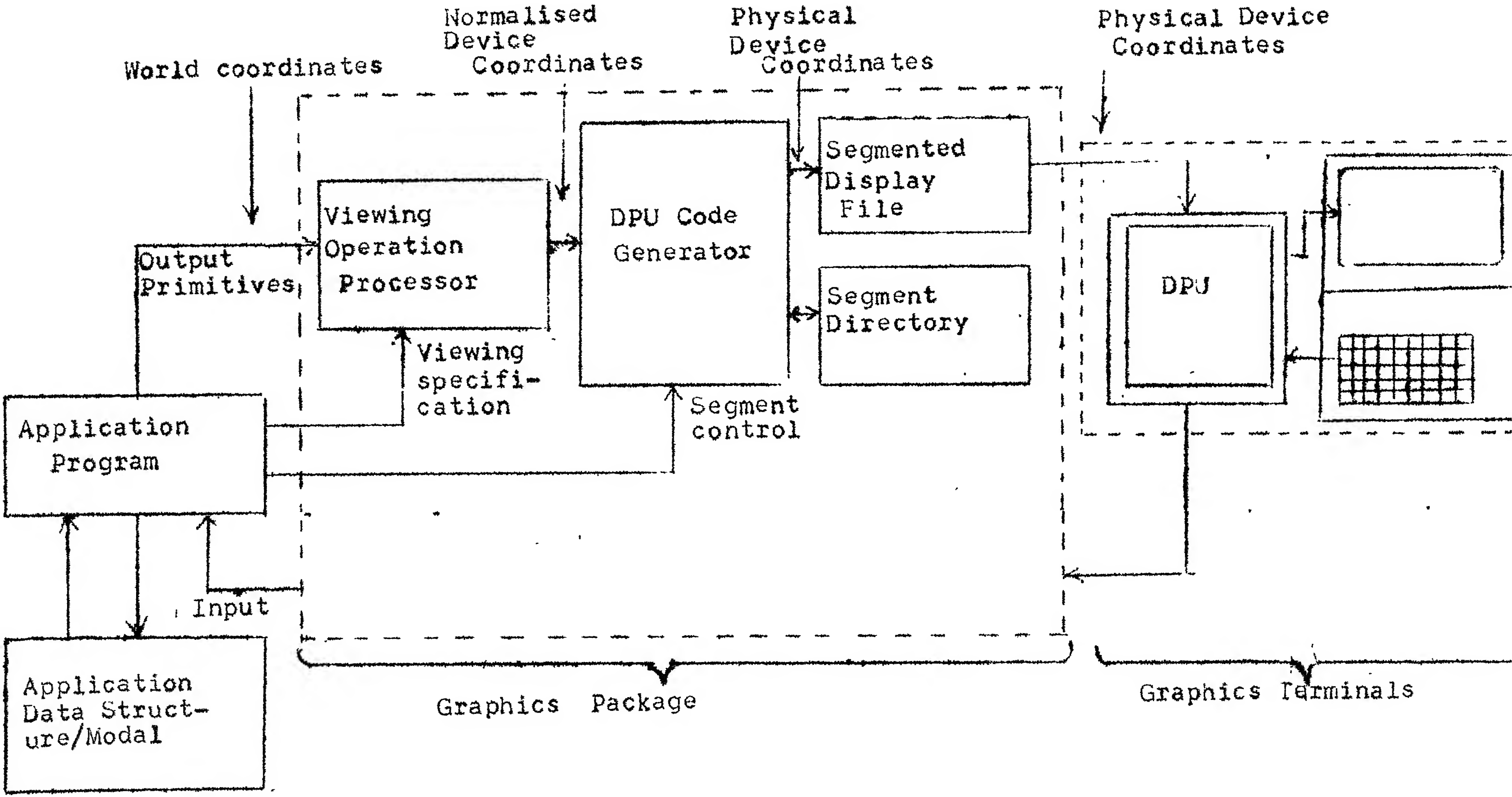


Fig. 1.1: Programmar's Model of Computer Graphics

components of the object, object attributes such as line style, color, or even surface texture and connectivity relationships and positioning data that define how the components fit together.

Having constructed an application model of the world as a set of one or more objects in an application data structure, the application programmer describes the model to the graphics system through calls to graphic primitives such as points, lines, polygons or character strings oriented in a three dimensional world. The application program must also specify to the graphics system what part of the object, seen from what vantage point, is to be displayed and on what part of the display surface, image should appear. The specification of viewing operation may be thought of as adjusting the settings of a synthetic camera. Other calls specify the division of the object into logical units i.e. segments.

The viewing operation processor uses the viewing specifications, to clip the object primitives against user supplied or default view volume boundaries, to project the object onto a plane surface and then to map the visible portion of the projection into the current viewport. Afterwards the output primitives are fed to the DPU code

generator DPU code generator, transforms the still device-independent specifications of the (clipped) primitives from normalised device co-ordinates into the device-dependent hardware instructions and device coordinates of the DPU. The segment functions control the segmentation of this DPU 'machine code', and specify to the 'DPU code generator' which segments are to be added, made visible/invisible, translated or deleted. DPU transforms the DPU program into the image which later is displayed on the screen.

1.2 SIMULA AND ITS FEATURES:

SIMULA 67 is a general purpose high-level programming language comparable in power to PL/I or ALGOL 68. SIMULA is based on ALGOL 60. With the addition of record oriented dynamic memory allocation, reference (pointer) structures, sets and queues , text and character handling, sequential and direct access input-output, quasi-parallel sequencing (co-routines) and process (event) oriented simulation capabilities. Well adapted to structured programming methodology, SIMULA 67 will often considerably reduce programming time compared to conventional languages like FORTRAN, COBOL, or PL/I. SIMULA 67 on DEC SYSTEM-10 contains two major additions to the SIMULA language: a system for separately

compiled program modules in SIMULA, FORTRAN, or MACRO-10 and a powerful online debugging system, SIMDDT. SIMULA compiles at half the speed of the DECSYSTEM-10 ALGOL compiler. The CPU time, when running SIMULA programmes, is about the same as for ALGOL, faster for input-output and text string handling, slower for stack oriented memory allocation.

1.3 STATE OF THE ART OF THE GRAPHICS PACKAGES:

Some of the most popular graphics packages are

1) GINO : Computer Aided Design Centre; Cambridge; England.
2) CALCOMP : California Computer Products Inc, Arrahelm
3) PLOT10 : Tektronix Inc; Beaverton.
4) IGS : Computer Centre; University of Michigan
5) TENEX E&S DISPLAY SOFTWARE : Bolt Beranek and Newman; Cambridge.
6) OMNIGRAPH : Xerox Palo Alto Research Centre.
7) GPGS : University of Nijmejen; The Netherlands
8) DISSPLA : Integrated Software System Corporation, California.

The packages IGS and TENEX E and S DISPLAY SOFTWARE are based on the use of structured display files.

The packages 'GINO, OMNIGRAPH , GPGS and DISPLA' offer some degree of device independence at application programming level.

With a view to make application programs portable, a standard called 'Core Graphics System' was developed by 'Graphic Standards Planning Committee' of ACM/SIGGRAPH in 1977 and it was refined in 1979.

There have been a number of efforts to extend standard high-level programming languages such as FORTRAN, PL/I, ALGOL 68 and PASCAL with graphics data types and operates to provide a more consistent and more elegant interface to graphics system than provided by a subroutine package.

1.4 OBJECTIVE AND SCOPE OF THE PRESENT WORK:

Many kinds of computer input and output are now a days programmed in standard ways, using high-level programming languages. For example, languages like PASCAL, SIMULA include facilities for file input and output and for handling interactive terminals. The ability to express such operations within a standard high-level language makes the programming much easier and permits the resulting programs to be run on a wide variety of different computers. We would like our graphics application programs to be equally easy to write and equally portable.

Now a days a large proportion of application programming is being done in SIMULA. Our aim is to provide programming interface for graphical input and output in SIMULA, so as to enable the application programmer to design and implement graphics application programs with ease and quickness. There are two approaches in providing this programming interface . One approach is the use of functions or procedures to access the capabilities of the graphics system. The other approach is to extend the programming language namely SIMULA with special statements and programming constructs for graphical input and output. For a device-independent graphics system, however, it is more appropriate to use a package of functions than a set of language extensions. The aim of device-independence is, afterall, to achieve portability and use of a special language leads to the need for special compilers that are unlikely to be plentifully available. So it was decided to develop 'graphics package' type programming interface. It consists of two parts. The first part called 'GSIMULA' deals with displaying images of two dimensional objects. The second part called 'GRAS' deals with displaying images of three dimensional objects. The facilities provided by GRAS are:

(1) Different types of planar geometric projections (2) Instant transformations (3) Picture segmentation. (4) Hidden line/surface elimination. The design of this package is based on the standards proposed by Graphic Standard Planning Committee of ACM/SIGGRAPH.

CHAPTER 2

SPECIFICATIONS OF GRAS

GRAS offers a compact but functionally complete command set to display the images of three dimensional objects. This command set can be divided into 5 distinct classes.

1. Output functions
2. Viewing functions
3. Instance transformation functions
4. Picture segmentation functions
5. Control, and other functions

2.1 OUTPUT FUNCTIONS:

An application programmer generates complete pictures by inserting calls to these functions in the application program.

Assume that an imaginary pen is tracing the three dimensional scene that is to be displayed. Movement of this imaginary pen is controlled by these functions. As the pen moves in the world coordinate space, the package transforms the coordinate data. The display device draws the picture of the scene on its screen using the transformed data. In the material that follows, we

often use the term 'CP'. It may be thought as a storage element and is used to store the position of the imaginary pen, obtained after any output primitive fed to it is effected [2].

A-MOVE-3 (x,y,z):

Where parameters 'x,y,z' are of REAL type and represent a point in world co-ordinate units.

This function is used when it is required to move the pen to the point (x,y,z). While moving, the pen does not mark on the scene. After the pen has reached the destination, the 'CP' is updated.

R-MOVE-3 (DX,DY,DZ):

Where parameters 'DX, DY, DZ' are of REAL type and these represent displacements in the x,y,z directions of the world coordinate space.

This function displaces the pen from 'CP' by DX, DY, DZ in x,y,z directions. While moving, the pen does not mark on the scene. After the pen has reached the destination, 'CP' is updated.

A-LINE-3 (x,y,z):

Where parameters 'x,y,z' are of REAL type and they represent a point in world coordinate space.

This function makes the pen to draw a line from the point denoted by 'CP' to the point at (x,y,z). After the pen has completed drawing the line, the position of the pen is saved in 'CP'.

R-LINE-3 (DX,DY,DZ):

Where parameters 'DX, DY, DZ' are of REAL type and they represent displacements in x,y,z directions of world co-ordinate space.

This function makes the pen to draw a line from point denoted by 'CP' through displacement by DX, DY, DZ. After the pen has completed drawing the line, 'CP' is updated.

A-POLYGON-3 (AX, AY, AZ, N):

Where parameters 'AX, AY, AZ' are arrays of REAL type. Parameter 'N' is of INTEGER type. Arrays AX, AY, AZ hold the co-ordinate data of the vertices of a polygon in world coordinate space.N represents the number of vertices in the polygon.

This function makes the imaginary pen to draw a polygon. The polygon is drawn first by moving to the first vertex, then drawing lines between consecutive vertices, and then finally closing the polygon by drawing a line from Nth vertex to first vertex. After the complete polygon is drawn, the 'CP' is updated.

R-POLYGON-3 (AX, AY, AZ, N):

Where parameters 'AX, AY, AZ' are arrays of REAL type. Parameter 'N' is of INTEGER type. Arrays AX, AY, AZ hold the displacements of vertices of a polygon in x,y,z, directions of world coordinate space. N represents the number of vertices in the polygon.

This function makes the pen to draw a polygon by displacing the pen successively 'N' times(by values given in arrays),and then finally closing the polygon. After polygon is completed, the 'CP' is updated.

Circle-3 (R, CX, CY, CZ):

Where parameters R, CX, CY, CZ are of REAL type. This function makes the pen to draw a circle of radius R with centre at (CX, CY, CZ) in the plane parallel to xy-plane of world coordinate space.

2.2 VIEWING FUNCTIONS:

In computer graphics dealing with the three dimensional scenes, the display surface being two dimensional each point in the scene must be mapped to some point on a plane surface (Fig. 2.1). Generally mapping of the scene onto a surface is also called projection. Conventionally, the point on the plane surface to which a point on the scene is

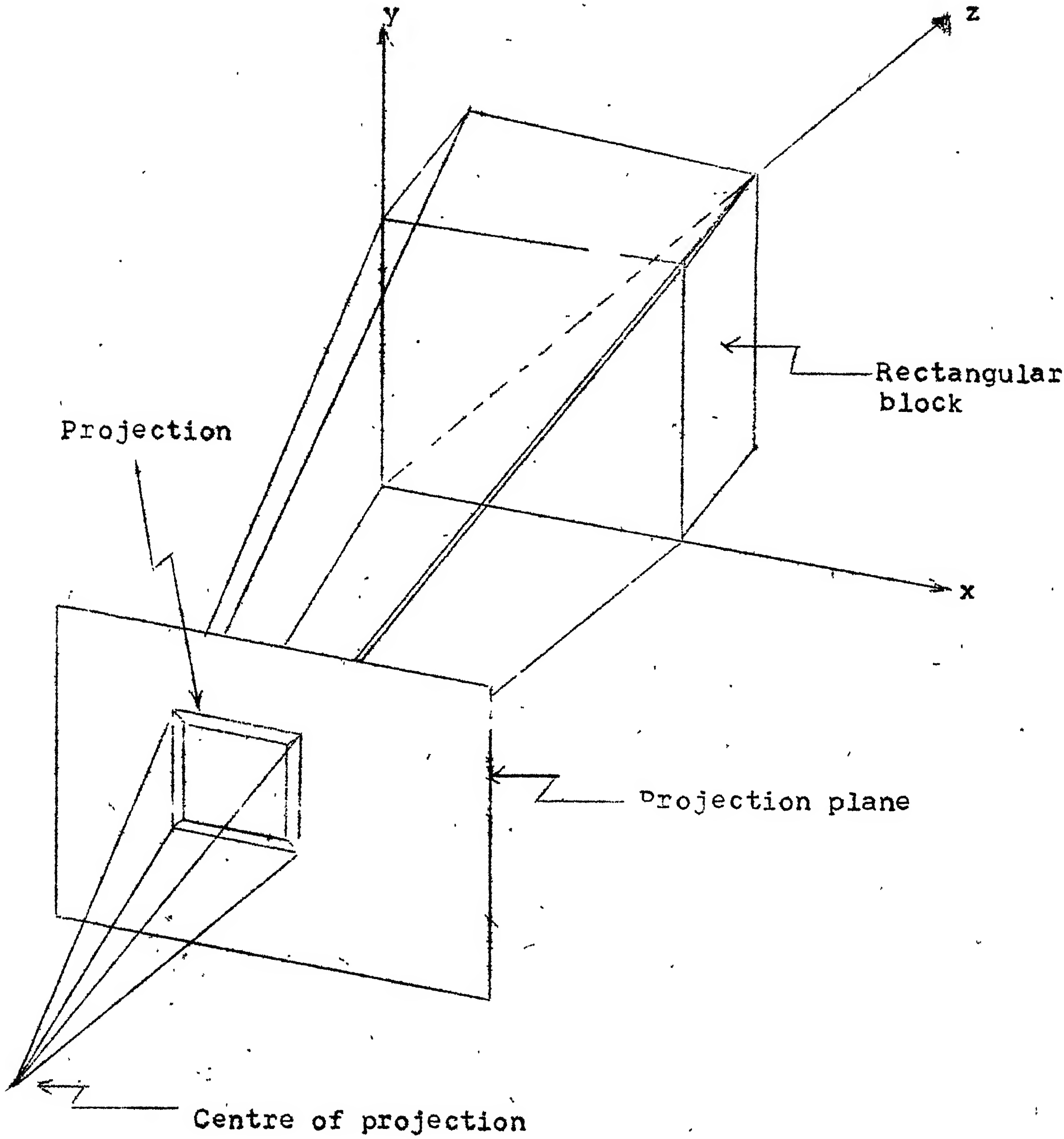


Fig. 2.1: Projection of rectangular block onto a plane.

projected, is obtained by finding the point of intersection of the line, joining the two points: (1) the point on the scene under consideration and (2) a particular point called 'centre of projection', with the plane surface. Then the projection of the scene is displayed on the screen of the display device. We designate the plane onto which the scene is projected by the term 'view plane'. A coordinate system incorporated on this plane to measure the coordinates of the mapped points is called 'U-V system'. [4].

Whenever a photographer wants to take a photograph of a real scene, he will do three things before clicking the camera: he selects a location for the camera, he sets the camera towards the scene, he adjusts the lens which determines how much of the scene will be included in the picture.

Analogous adjustments are done by parameter specifications of the viewing functions.

<u>Set-up-view plane (RX, RY, RZ, NX, NY,NZ,D, PX,PY,PZ)</u>: Where parameters 'RX, RY, RZ, NX, NY, NZ, D, PX, PY, PZ' are of REAL type.

This function determines location and orientation of the view plane onto which the projection of the scene is taken. Assume that a telescopic stick is attached

orthogonally to the view plane at the origin of U-V system and the free end of stick is positioned at a point called 'Reference point'. By adjusting the length of the stick and rotating the stick about the reference point, the location and orientation of view plane can be altered. By choosing the reference point near the centre of object to be viewed, views of the object from different directions can be obtained easily. Parameters 'RX, RY, RZ' represent the co-ordinates of reference point in world coordinate units. Parameters 'NX, NY, NZ' represent the co-ordinates of some point on the stick relative to the reference point. Parameter 'D' represent the distance of view plane from reference point. Rotating the camera about viewing direction shows the same scene but puts a different part of the object up. Similarly any part of the object can be made vertical in the image, by the specification of a vector (PX, PY, PZ). The projection of this vector is taken onto the view plane, and the view plane is rotated till this projection coincides with the V-axis.

Set-projection-Parameters (VX, VY, VZ, Flag):

Where parameters 'VX, VY, VZ' are of REAL type and parameter 'Flag' is of BOOLEAN type. when the centre of projection is at a finite distance, from the object, then

the projection is called perspective projection and when it is far away, then all the lines joining the centre of projection become parallel, and the projection is called parallel projection.When application programmer intends to obtain perspective projection, he should assign the 'Flag' to TRUE and parameters 'VX, VY, VZ' to the coordinates of 'centre of projection' in U-V-W co-ordinate space. U-V-W co-ordinate system is a right-handed system and is obtained by extending U-V-system with W-axis perpendicular to view - plane. When the application programmer intends to obtain parallel projection then he should assign 'Flag' to FALSE and parameters 'VX, VY, VZ' indicate direction of projection in U-V-W system.

Set-Window (XL, XH, YL, YH):

Where parameters XL, XH, YL, YH are of REAL type. The object to be viewed is projected onto the view plane. The graphicspackage must be informed, what portion of the plane the application programmer intends to display, for it has to transform output primitives from U-V system to screen coordinate system. The programmer should specify only rectangular portions with edges parallel to U-V coordinates. The parameters 'XL, XH, YL, YH' define the low and high limits of window along each coordinate axis of the U-V system.

Set-View-Depth (FD, RD)

Where the parameters 'FD, RD' are of REAL type. Specifications of view plane and type of projection determine view volume. (Fig. 2.2 and Fig. 2.3). In case of perspective projection, the 'view volume' is semi-infinite pyramid whereas, in the case of parallel projection it is infinite parallelepiped. In either case, they may be truncated to a finite view volume by specifying a front clipping plane and a back clipping plane. Both clipping planes are parallel to the view plane and both are specified by distances along the view plane-normal FD, RD specify these distances.

Truncation of view volume is desirable when the user wants to view only a certain portion of view volume, eliminating the extraneous information, outside the desired view volume. For perspective projections, there is an additional motivation. A very distant object comprising many output primitives may appear on the view surface as a 'Blob' with no distinguishable form. Also an object which is very near to the view point may extend across the entire viewport like so many pick up sticks conveying useless information to the viewer and in fact distracting from the perception of useful information.

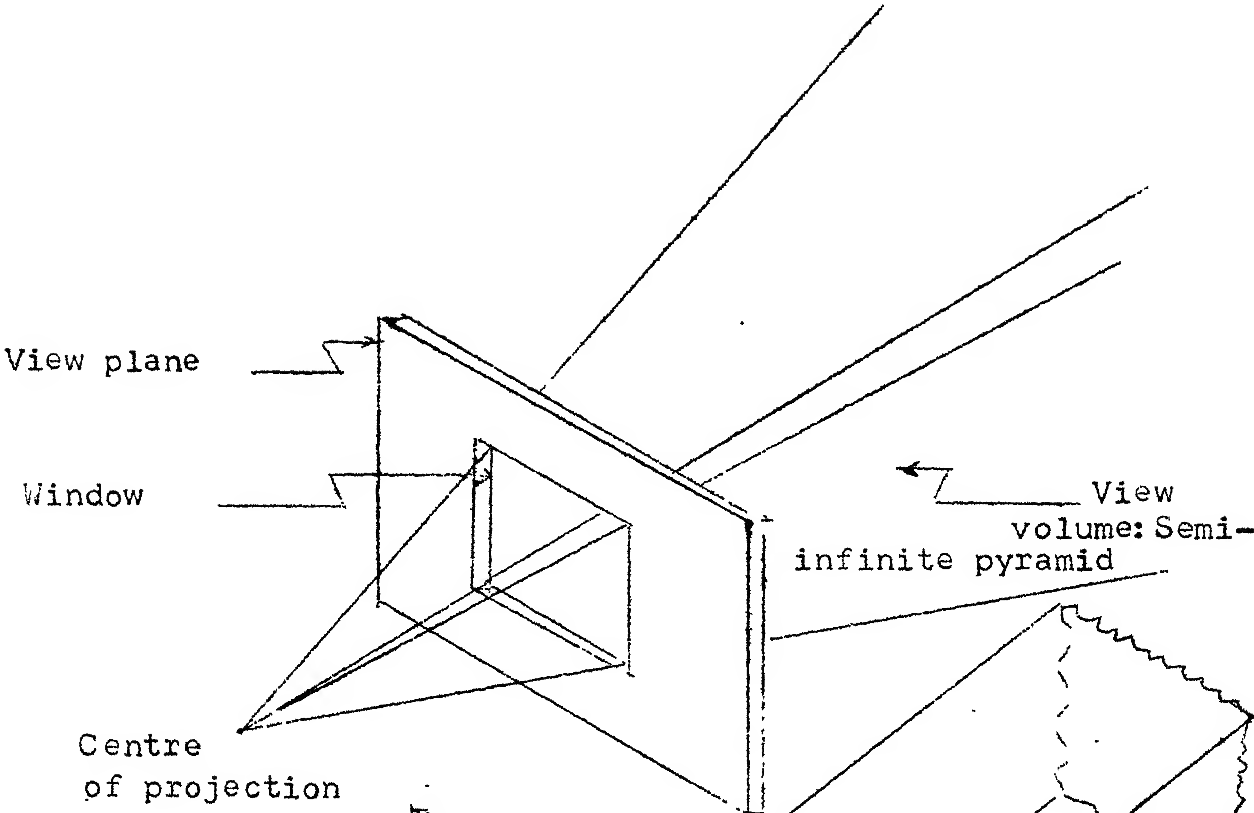


Fig. 2.2: Perspective projection

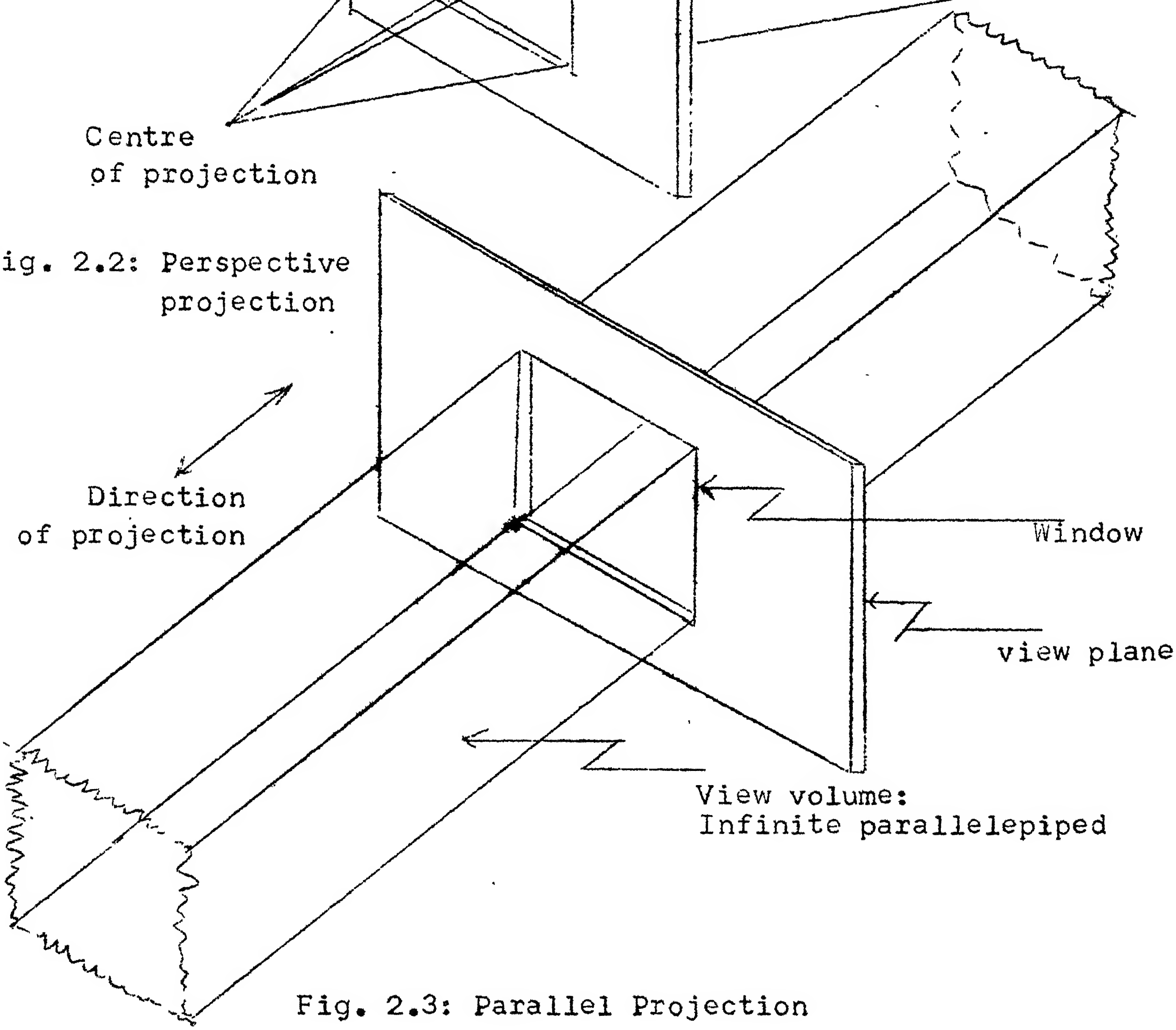


Fig. 2.3: Parallel Projection

Set-depth-Flags (Flag1 Flag2):

Where parameters 'Flag 1, Flag2 ' are of BOOLEAN type. Flag1 when TRUE, enables clipping against front clipping plane

Flag2 when TRUE enables clipping against back clipping plane.

Set-viewport (XL, XH, YL, YH):

Where parameters 'XL , XH, YL, YH' are of INTEGER type. A viewport is a rectangle on the screen where the programmer would like th window's contents displayed. It is often useful to specify a viewport smaller than screen, for we can then leave room for command menus, system messages, and also we can display different views and projections onto different parts of the screen. The parameters 'XL, XH, YL, YH' define low and high limits of viewport along each axis of the screen co-ordinate system. The range of values for XL and XH is from 0 to 1024 where as for YL and YH, it is from 0 to 750.

2.3 INSTANCE TRANSFORMATION FUNCTIONS:

The geometric models and other forms of data that are to be displayed, often have a clearly evident structure. They include structures which are repeated. Instance transformation functions allow these structures to be placed

any where in the world coordinate system in any orientation and size by geometric transformations: translation, rotation, scaling.

In charts we find definitions of graphical symbols in the key of the chart and instances of each graphical symbol at different locations. Similarly we define symbols by a sequence of output primitive calls in a different coordinate system called 'Master coordinate system' and these symbols are positioned by making calls to transformations prior to the symbol call [1].

Scale-3 (SX,SY,SZ): This function scales the symbol relative to the origin by factors Sx,Sy and Sz in the x,y, and z directions respectively.

Rotate (I, Phi) : This function rotates the symbol anticlockwise through Phi Degrees

I=1 implies the rotation is with respect to x-axis.

I=2 implies the rotation is with respect to y-axis

I=3 implies the rotation is with respect to z-axis.

Translate-3(Tx,Ty,Tz): Translates the symbol through distances Tx,Ty and Tz measured in the x,y and z directions respectively.

Above transformations, when called, post multiplies the instant transformation matrix.

Begin-tran: This function when called does the following operation.

1) Pushes the overall transformation matrix on to the stack.
2) Multiplies instant transformation matrix with overall transformation matrix.
3) Makes instant transformation matrix to identity.

End-tran: Restores the overall transformation matrix from the stack.

2.4 SEGMENTATION FUNCTIONS:

The image on the display screen is often composed of several pictures or items of information. We might wish to show all the information simultaneously or at other times look at individual items. This can be done with the help of picture segmentation functions. The package provides functions to name different parts of the picture and to effect the desired modifications on these parts [4].

Create-segment (seg-name): This function starts a new segment and this segment is named by the parameter passed through this functiona call. Output functions, called subsequent to this call are added to this segment.

Close-segment: When application programmer has added instructions to a segment and wants to add no more instructions, then he can do so by calling this function.

Delete-segment (seg-name): If a segment is no longer required and its storage is to be recovered, it can be done by calling this function.

Delete-all segments: This function deletes all segments.

Post-segment (seg-name): Visibility of the segment named with th with the supplied parameter is set to 'VISIBLE'.

Unpost-segment (seg-name): Visibility of the segment named with the supplied parameter is set to 'INVISIBLE'

Update-display: Transmits DPU code belonging to the visible segments to the display processor.

2.5 CONTROL AND OTHER FUNCTIONS:

Clear-screen: This function clears the screen of the display device.

Draw-viewport: This function draws the specified viewport on the screen of the display device.

Init-segments: This function initialises the segment directory and the buffers.

Eliminate-hiding: Lines or parts of lines obscured by the surface of the same object or other objects are eliminated.

CHAPTER 3

IMPLEMENTATION DETAILS

The sequence of processes through which the output primitives undergo before they are displayed is shown in the Fig. 3.1.

Application data structure holds data of the object to be viewed. Application program issues output primitive calls which describe the object. These output primitives undergo a sequence of processes before they are displayed.

3.1 INSTANCE TRANSFORMATION:

Each output permitive is transformed with overall transformation matrix. Initially the overall transformation matrix and instant transformation matrix are set to identity.

Each transformation call postmultiplies instant transformation matrix. After specifying complete transformation and prior to the symbol calling, the function 'Begin-Tran' is called. This function does the following operations: 1) Overall transformation matrix is saved on the stack, 2) instant transformation matrix is multiplied with overall transformation matrix and 3) instant transformation matrix is set to identity. Then the symbol

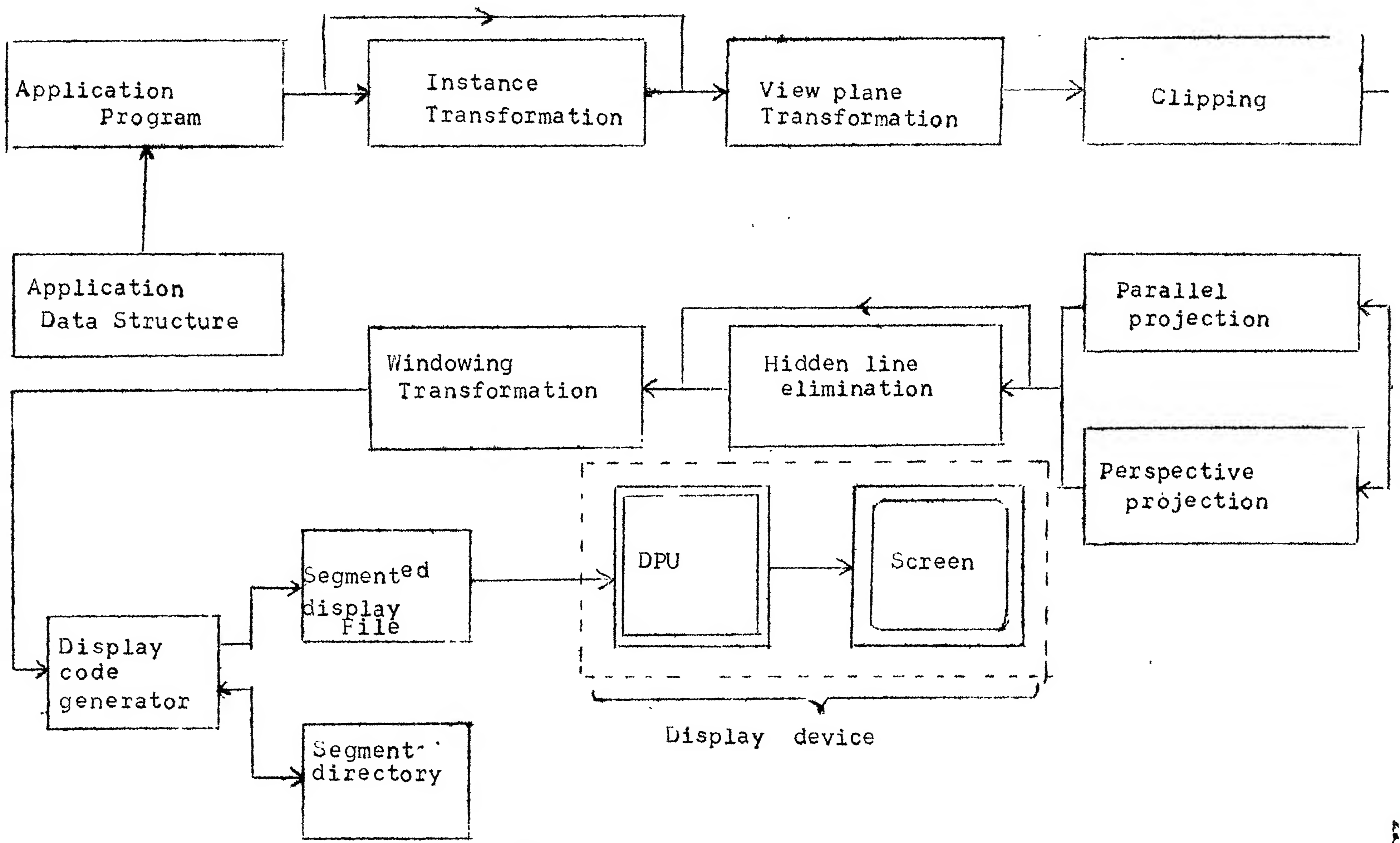


Fig. 3.1: Processing of output primitives.

is called after which overall transformation matrix is reset with that on the top of the stack.

3.2 VIEW-PLANE TRANSFORMATION:

The application programmer defines the object/ objects to be viewed in world coordinate units. The objects are projected onto the view plane. Then this projected view is displayed on the screen of the display device. Computation of projection will be easier if the object description is transformed into a coordinate system lying in the view plane. U-V-W system is a right-handed system with the origin stationed at the point where the 'view plane normal' pierces the view plane. Its U and V-axes lie in the view plane. W-axis is perpendicular to the view plane. Transformation of the object specified in world coordinate units to view plane coordinate units is done by multiplication of co-ordinates of the points with a matrix and this process is called view plane transformation. The matrix that is used in this transformation is generated by the procedure ' setup-view plane'.

The transformation matrix which when applied to word coordinates of a point, yields U-V-W coordinates of the point, can be obtained by combining primitive transformations. These primitive transformations correspond to different

stages in positioning the U-V-W system. The first step is positioning the origin of the U-V-W system at the point where view-plane normal pierces the view plane such that its axes are parallel to x,y,z axes of the world coordinate system. After the origin is in place, the w-axis is aligned with the view plane normal. This can be done by rotating U-V-W system first about U-axis until view plane normal is in u-w plane and then about v-axis until w-axis coincides with the view plane normal. Now the only step remaining is to rotate the U-V-W system about w-axis until the projection of the vector (PX, PY, PZ) on to view plane coincides with V-axis. The entire transformation sequence is given by (T) * (Ru) * (Rv) * (Rw).

where,

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Tx & -Ty & -Tz & 1 \end{bmatrix}$$

$Tx = (Rx) + (D * Nx)$

$Ty = (Ry) + (D * Ny)$

$Tz = (Rz) + (D * Nz)$

Tx,Ty,Tz = coordinates of the point where the view plane normal pierces the view plane

Rx,Ry,Rz = co-ordinates of reference point

Nx,Ny,Nz = coordinates of normalised view plane vector

D = distance of view plane

$$Ru = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta u) & \sin(\theta u) & 0 \\ 0 & -\sin(\theta u) & \cos(\theta u) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

θu = Angle through which the u-v-w system is rotated clock-wise about u-axis

$$Temp = \sqrt{(Ny)^2+(Nz)^2}$$

$\sin(\theta u)$ = Ny/temp; $\cos(\theta u)$ = Nz/temp.

$$Rv = \begin{bmatrix} \cos(\theta v) & 0 & -\sin(\theta v) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta v) & 0 & \cos(\theta v) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

θv = Angle through which u-v-w system is rotated clockwise about v-axis.

$\sin(\theta v) = -Nx$

$\cos(\theta v) = temp.$

$$Rw = \begin{bmatrix} \cos(\theta w) & \sin(\theta w) & 0 & 0 \\ -\sin(\theta w) & \cos(\theta w) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

θw = angle through which u-v-w-system is rotated clockwise about w-axis so that the projection of positioning vector (Px, Py, Pz) coincides with v-axis.

PRx = length of the projection of position vector along u-axis.

PRy = length of the projection of position vector along v-axis.

$$Save = \sqrt{(PRx)^2 + (PRy)^2}$$

$$\sin(\theta w) = PRx/save$$

$$\cos(\theta w) = PRy/save$$

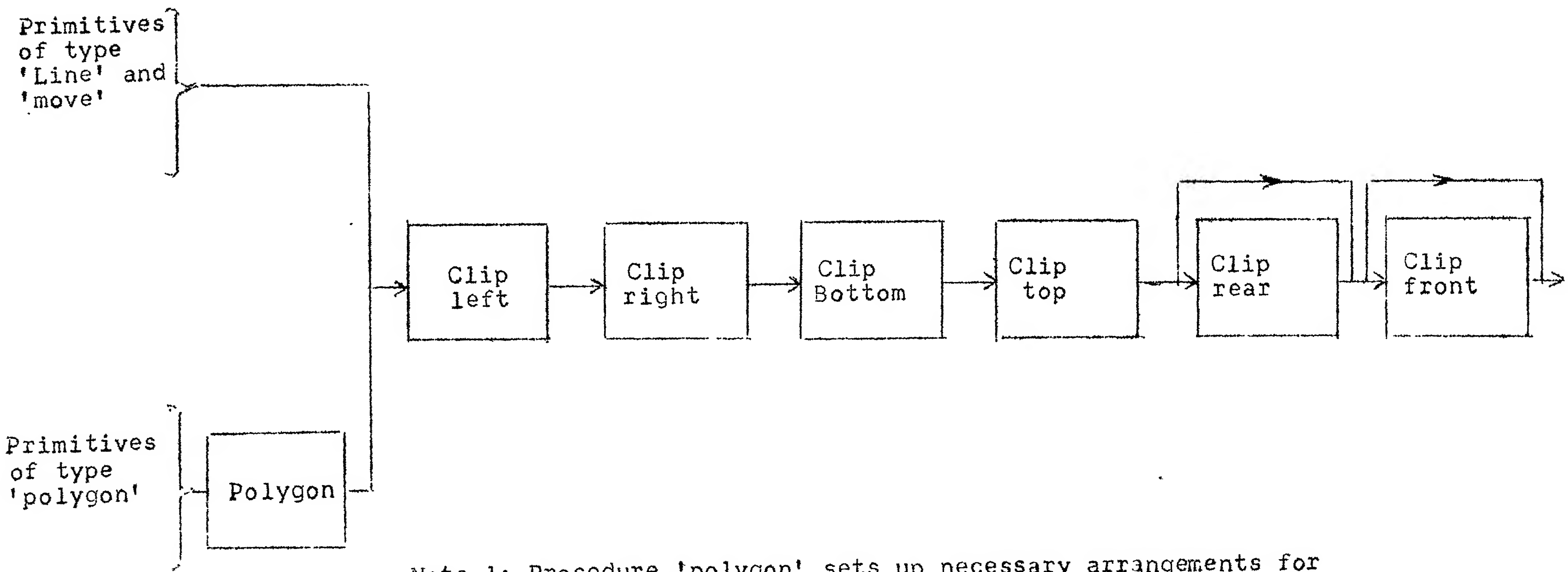
3.3 CLIPPING:

In this process the portion of the three dimensional scene that is within the view volume is displayed and the rest is discarded. There are two reasons why clipping is desired. The first reason is, when one wants to display

a small portion of a large scene, transformed output primitives of the scene lying outside the size of the display device may cause problems such as 'wraparound' due to the overflow of internal coordinate registers of the display device. The second reason is to avoid processing of undesired portion by processes ahead. The view volume is determined by window and projection specifications.

There are several clipping algorithms available. The algorithms, we used, is polygon clipping algorithm discovered by Sutherland and Hodgman. This algorithm clips lines as well as polygons. Polygons when clipped with this algorithm, remain polygonal. Simple line clipping algorithms will clip polygons to outlines that are no longerclosed. To close the outline, appropriate sections of window boundary must be added to it.

Our clipping algorithm is based on the idea that it is relatively easy to clip a polygon against a single clipping plane rather complete view volume. Complete clipping is performed by clipping the polygon successively by all the six planes of the view volume. The output from each clipping stage is a polygon i.e. an ordered sequence of vertices. As the order of generation of vertices by a clipping plane is same as the order of feeding to the next clipping plane, it is possible to begin clipping on second bounding plane before clipping of the entire figure



Note 1: Procedure 'polygon' sets up necessary arrangements for clipping the polygon.

Note 2: Processing by each clipping plane is shown in the next page.

Fig. 3.2: Block diagram of clipping process.

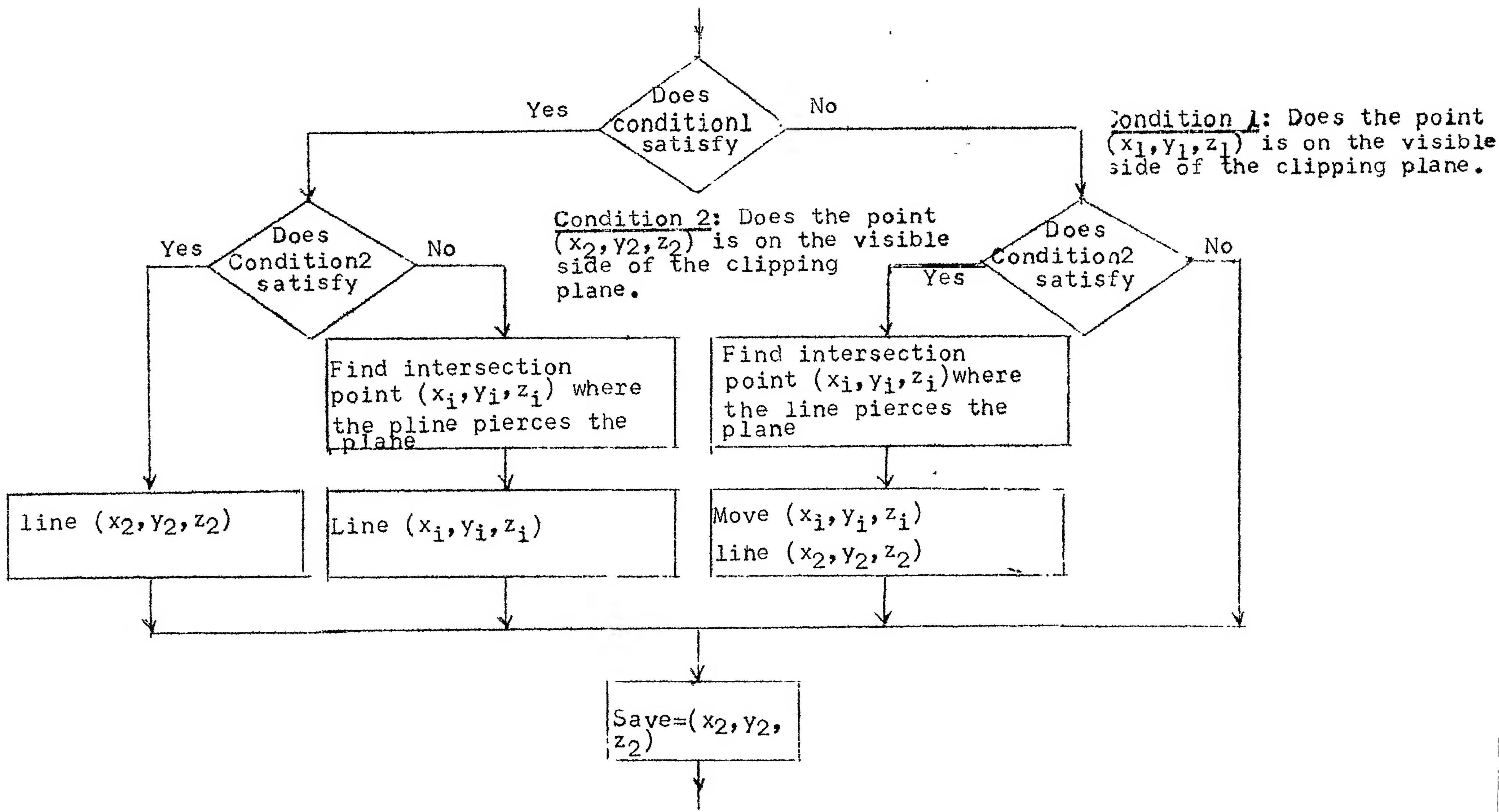


Fig. 3.3: Processing by each clipping plane.

against the first boundary plane is completed. The block diagram of clipping process is given in Fig. 3.2.

3.4 PARALLEL PROJECTION:

A parallel projection is formed by extending parallel lines from each of the vertices on the object until they intersect the view plane. The point of intersection is the projection of vertex. We connect the projected vertices by line segments which correspond to connections of the original object.

Determination of Projection of a Point on to the xy-Plane:

Suppose that the direction of projection is given by the vector $[X_p, Y_p, Z_p]$ and it is required to find the projection of a point on the object at (x_1, y_1, z_1).

$$x = x_1 + (x_p)u \qquad (3.4.1)$$

$$y = y_1 + (y_p)u \qquad (3.4.2)$$

$$z = z_1 + (z_p)u \qquad (3.4.3)$$

The z-cordinate of any point on xy-plane is equal to 0. Therefore point of intersection of the line with xy-plane can be obtained by solving equations (3.4.1), (3.4.2) and (3.4.3) with $z = 0$.

Solving equation (3.4.3) for u we get

$$u = -\frac{z_1}{z_p} \qquad (3.4.4)$$

substituting this value of u into equations (3.4.1) and (3.4.2) we get x and y co-ordinates of intersection point,

$$x = x_1 - z_1 \left(\frac{x_p}{z_p}\right) \qquad (3.4.5)$$

$$y = y_1 - z_1 \left(\frac{y_p}{z_p}\right) \qquad (3.4.6)$$

3.5 PERSPECTIVE PROJECTION:

The perspective projection of an object is formed by passing lines (projectors) from a 'centre of projection' through each point on the object and finding projectors' intersection with the view plane.

<u>Determination of Projection of a Point on to the xy Plane:</u>

Suppose that centre of projection is at (x_c, y_c, z_c) and it is required to find the projection of a point on the object at (x_1, y_1, z_1). Then the parametric equations for the line containing these two points are given by

$$x = x_c + (x_1 - x_c)u \qquad (3.5.1)$$

$$y = y_c + (y_1 - y_c)u \qquad (3.5.2)$$

$$z = z_c + (z_1 - z_c)u \qquad (3.5.3)$$

The z-coordinate of any point on xy-plane is equal to '0'. Therefore point of intersection of the line with xy-plane is obtained first solving the equation (3.5.3) for u, by substituting z = 0 and then substituting the value of u in equations (3.5.1) and (3.5.2).

Solving equation (3.5.3) for u

$$u = -\left(\frac{z_c}{z_1 - z_c}\right) \qquad (3.5.4)$$

Substituting this value of u into equation (3.5.1) and (3.5.2) we get x and y coordinates of intersection point

$$x = x_c - z_c\left(\frac{x_1 - x_c}{z_1 - z_c}\right) \qquad (3.5.5)$$

$$y = y_c - z_c\left(\frac{y_1 - y_c}{z_1 - z_c}\right) \qquad (3.5.6)$$

Equations (3.5.5) and (3.5.6) can be rewritten as

$$x = \frac{(x_c)(z_1) - (z_c)(x_1)}{(z_1 - z_c)} \qquad (3.5.7)$$

$$y = \frac{(y_c)(z_1) - (z_c)(y_1)}{(z_1 - z_c)} \qquad (3.5.8)$$

3.6 HIDDEN LINE ELIMINATION:

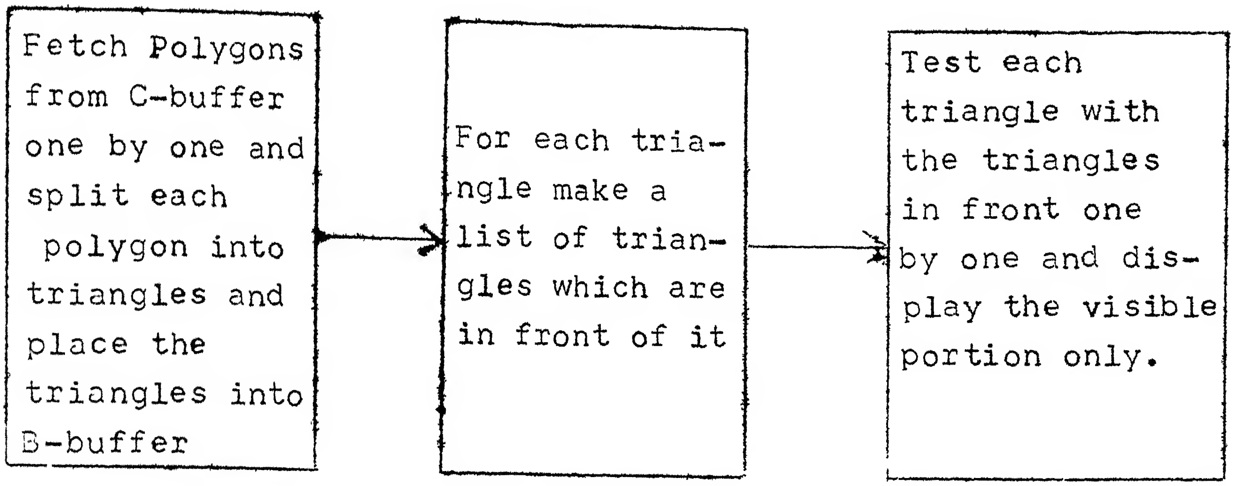
When an object is viewed, its entire surface is not visible. Only some parts of the surface are visible. What is visible and what is invisible depends upon the point of view. An area on the object becomes invisible, only when the light rays from it cannot reach the view point. The opaque material of the object makes certain areas invisible by preventing light rays from these areas in reaching the view point. In computer generation of an image, no such automatic elimination takes place when the objects are projected onto the screen coordinate system. Instead, all parts of the object including many parts that should be invisible, are displayed. A process must be employed which differentiates the visible portions and invisible portions. This process is known as hidden line/surface elimination.

There are a number of hidden line/surface algorithms - some simple, some very sophisticated. Some only work on special, peculiarly defined objects, whereas others work on all combinations of objects. These general algorithms are therefore complicated; moreover they may be limited by computer time and storage restrictions. We describe two methods here. The first is relatively simple while the second uses a general algorithm.

The first algorithm is called Back-Face detection and Removal algorithm and has to be used only when the order of vertices listed for each polygon corresponds to anti-clockwise order as seen from the outside of the object. This algorithm eliminates all hidden surfaces for a single convex body. The principle of this algorithm is the determination of surfaces which face the view point and identifying these faces as visible surfaces. A surface will face view point if the angle between the outward normal of the surface and viewing direction is between 0 degrees and 90 degrees.

The second algorithm is called Painter's algorithm. While the previous algorithm can remove many of the hidden lines and surfaces, it does not completely solve the problem. If we have two separate objects, then a front face of one object may obscure a front face of the second object. In Painter's algorithm, we can not process each polygon independently, as was done in the first algorithm. We must compare each polygon with all the rest to,see which is in front of which. The block diagram of Painter's algorithm is given on the next page.

The object is described to the system as a collection of n polygonal faces.



BLOCK DIAGRAM OF PAINTER'S ALGORITHM

Step 1: This step is performed by the procedure 'Split-into-triangles'. Polygons are split into triangles.

Step 2: is performed by the procedure 'Compare-all-triangles'. This procedure uses the function 'compare-two-triangles' which output '0' if the two triangles do not overlap. If the two triangles overlap then it outputs -1 when the first triangle is infront of the second and it outputs +1 when the first triangle is at the back of the second.

Step 3: Testing the entire triangle is done by testing the sides of the triangle one at a time with the triangles in front.

3.7 WINDOWING TRANSFORMATION:

The application programmer specifies a rectangle of interest in the view plane co-ordinate system and a rectangle on the screen of the display device. Let us suppose that edges of the window are

$$u = WXL$$
$$u = WXH$$
$$v = WYL$$
$$v = WYH$$

and edges of screen are

$$x = VXL$$
$$x = VXH$$
$$y = VYL$$
$$y = YYH$$

The graphicspackage transforms contents of window into the viewport.

The corners of window are mapped to corners of view-port. Let a point (XW, YW) in the window be mapped to a point (XV, YV) in the viewport.

$$(VXH - VXL) \propto (WXH - WXL) \qquad (3.7.1)$$

$$(XV - VXL) \propto (XW - WXL) \qquad (3.7.2)$$

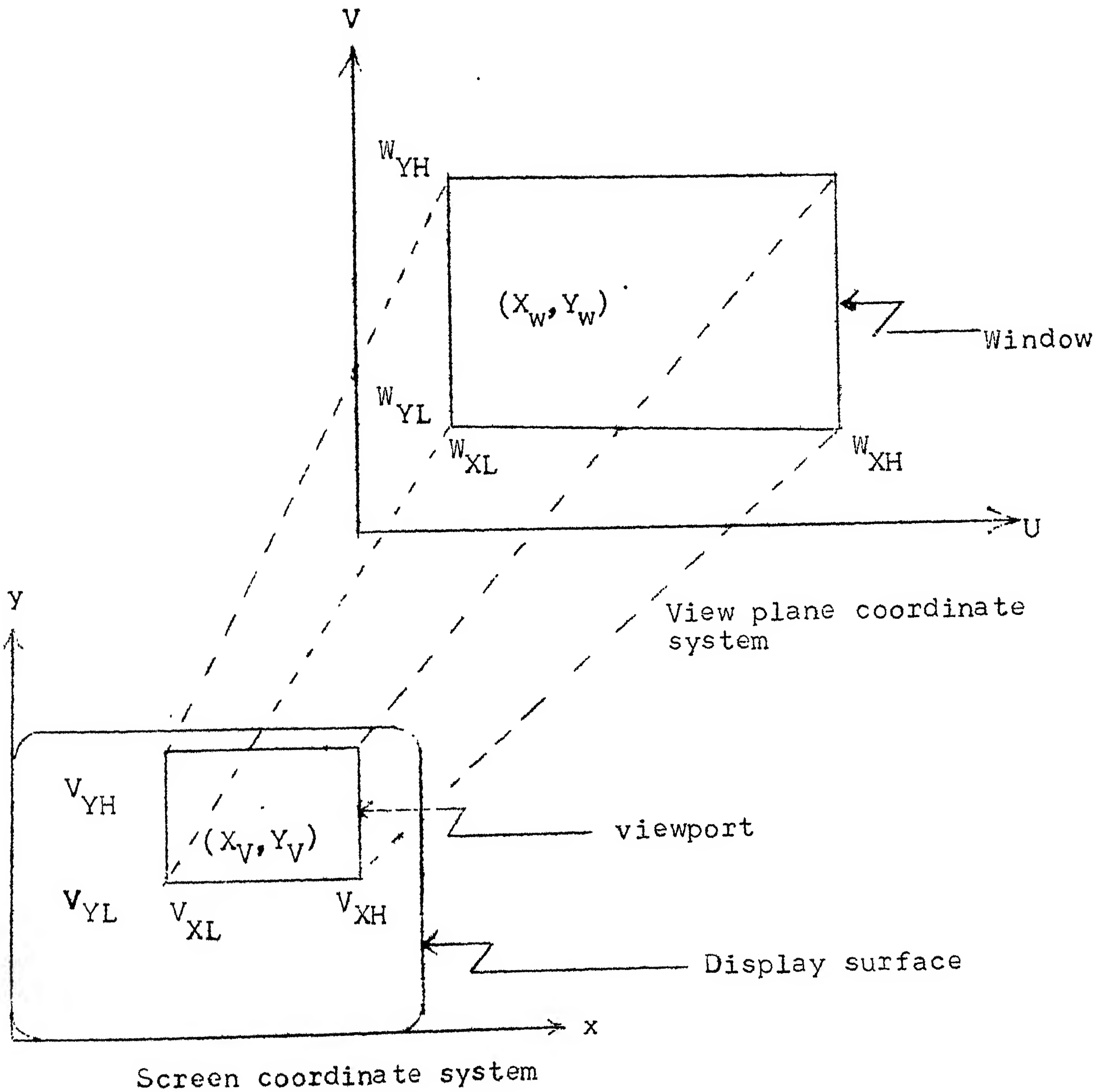


Fig. 3.7.1: Windowing Transformation.

Dividing eq. (3.7.2) by Eq. (3.7.1)

$$\frac{XV - VXL}{VXH-VXL} = \frac{XW - WXL}{WXH-WXL} \tag{3.7.3}$$

Rewriting Eq. (3.7.3)

$$XV = \left(\frac{VXH-VXL}{WXH-WXL}\right) (XW-WXL) + VXL \tag{3.7.4}$$

Similarly,

$$(VYH-VYL) \ \alpha \ (WYH - WYL) \tag{3.7.5}$$

$$(YV - VYL) \ \alpha \ (YW - WYL) \tag{3.7.6}$$

Dividing eq. (3.7.6) by eq. (3.7.5)

$$\frac{YV - VYL}{VYH-VYL} = \frac{YW-WYL}{WYH-WYL} \tag{3.7.7}$$

Rewriting eq. (3.7.7)

$$YV = \left(\frac{VYH - VYL}{WYH - WYL} \right) (YW-WYL) + VYL \tag{3.7.8}$$

<u>Display code generator:</u> converts device independent output primitives in NDC space into corresponding display instructions.

<u>Segmented Display file</u>: contains a list of display instructions generated by computer.

<u>DPU</u> : generally includes a vector generator and a character generator, which converts the display instructions into signals suitable for the display's deflection system.

CHAPTER 4

CASE STUDIES

During the development of the package, to debug and check the performance of the package, programming examples were developed. These examples show the typical usage of the command set [3].

4.1 SPHERE:

This program draws the figure of a sphere and is available in the file 'Sphere.Sim'. The photograph of the picture drawn by this program is shown in Fig. 4.1. The program generates a sphere by rotating a semicircle in yz-plane through 360 degrees about y-axis. The program describes the sphere to the graphics system by a sequence of calls to the primitive 'polygon'.

The semicircle is divided into 24 equal parts. The end points of the line segments are stored in arrays y_{in} and z_{in}. Initially these points are also stored in arrays x-pres, y-pres, z-pres. The semicircle is rotated through and angle deltita and now the co-ordinates of the points on the semicircle are computed from the co-ordinates stored in the arrays y_{in} and z_{in}. The computed co-ordinates are stored in the arrays x-next, y-next, z-next. Array set

'x-pres, y-pres, z-pres' and Array set 'x-next, y-next, z-next contain vertices for a strip of polygons and calls to polygons are issued. Then the coordinates of points in the set 'x-next, y-next, z-next' are transfered to the array set 'x-pres, y-pres, z-pres' and the array set 'x-next, y-next, z-next' is filled with the points got by rotating the semicircle through an angle deltita. Calls to polygons are issued and the process is repeated till the full sphere is drawn.

4.2 RING BALL:

This program displays a ring ball and the program for this is available in the file named 'RING.SIM'. The photograph of the picture drawn by this program is shown in Fig. 4.2. The ring is generated by rotating a circle positioned in YZ-plane with centre on the z axis at a distance of R from the origin, though 360 degrees. The program describes the object (i.e. Ring ball) to the graphics system by a sequence of calls to the output permitive 'polygon', and is similar to the program for sphere.

4.3 TYRE:

This program displays a cycle tyre and the program is available in the file named 'Tyre.Sim' . The photograph of the picture drawn by this program is shown in the Fig. 4.3.

The tyre is generated by rotating an arc of the circle through 360 degrees about y-axis and the program is similar to that of the ring.

4.4 HOUSING COMPLEX:

This program displays four houses and the program is available in the file 'Houses.Sim'. The photograph of the picture drawn by this program is shown in Fig. 4.4. This program reads data necessary for drawing a house from the file 'Houses Dat' . It generates 4 houses by applying instant transformations to the symbol 'House'. Each house is stored in a separate segment. Any combination of the houses can be displayed with this program.

4.5 HOLLOW CUBE:

This program displays a hollow cube and the program is available in the file 'Hollow.Sim'. The photograph of the picture drawn by this program is shown in Fig. 4.5. This program reads data for a single solid cube. Then it builds the hollow cube by instant transformations of solid cube. User can specify the viewing direction interactively and the program displays the view of the hollow cube in that direction.

4.6 PROJECTIONS:

This program displays various types of projections of a cube and the program for this is available in the file 'Demo.Sim'. The photograph of the picture drawn by this program is shown in Figs. 4.6 and 4.7. It illustrates one point, two point, three point perspective views and isometric, cabinet and cavalier projections.

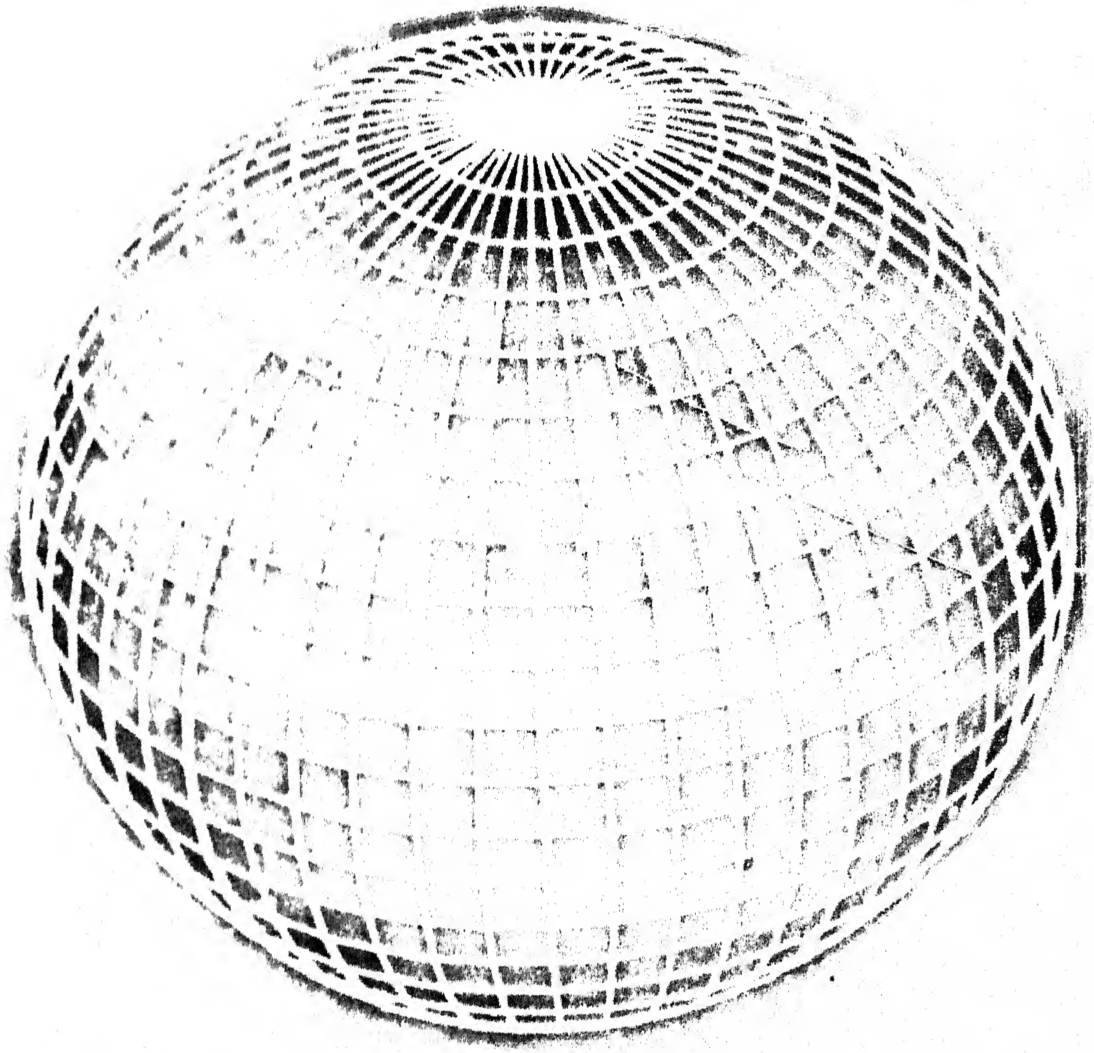


Fig. 4.1: SPHERE

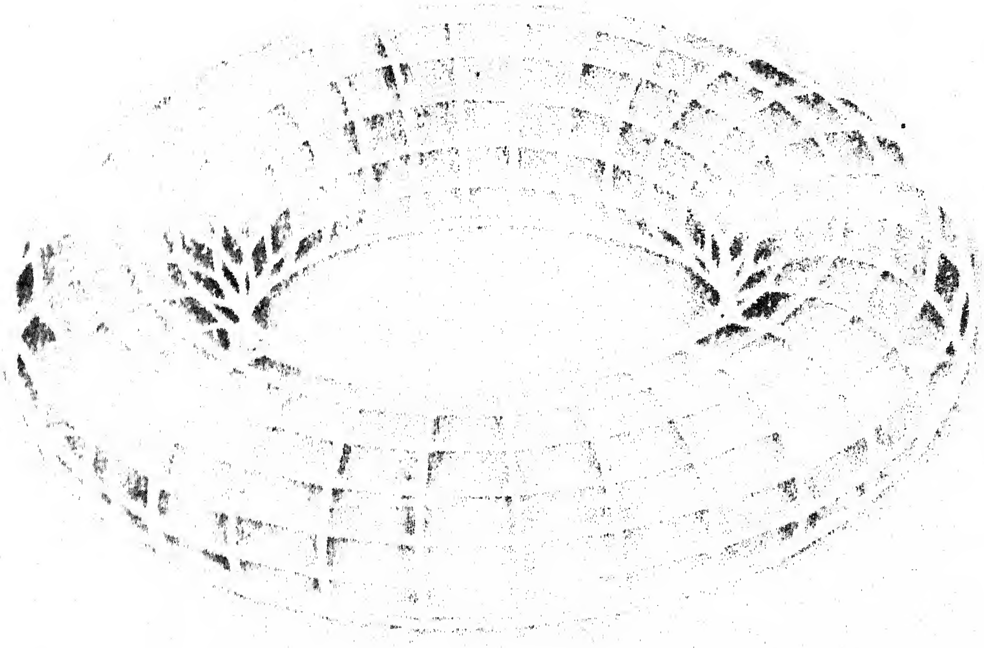


Fig. 4.2: RING BALL

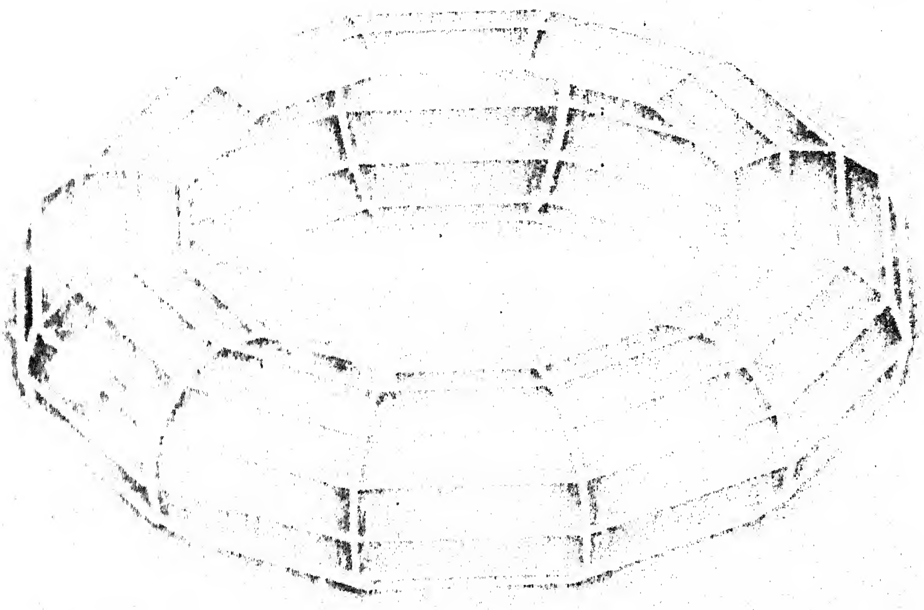


Fig. 4.3: CYCLE TYRE

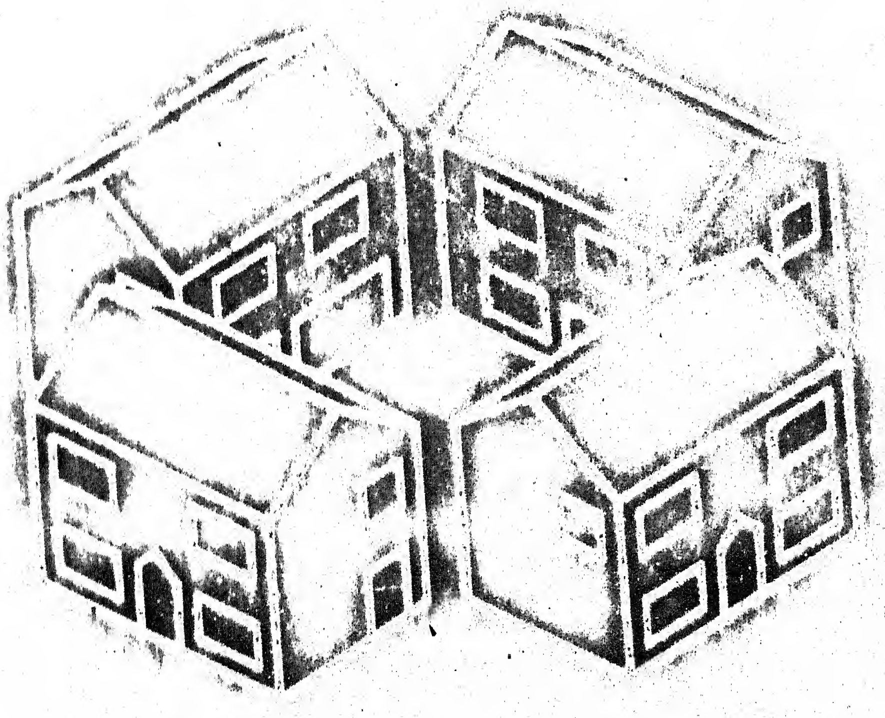


Fig. 4.4: HOUSING COMPLEX

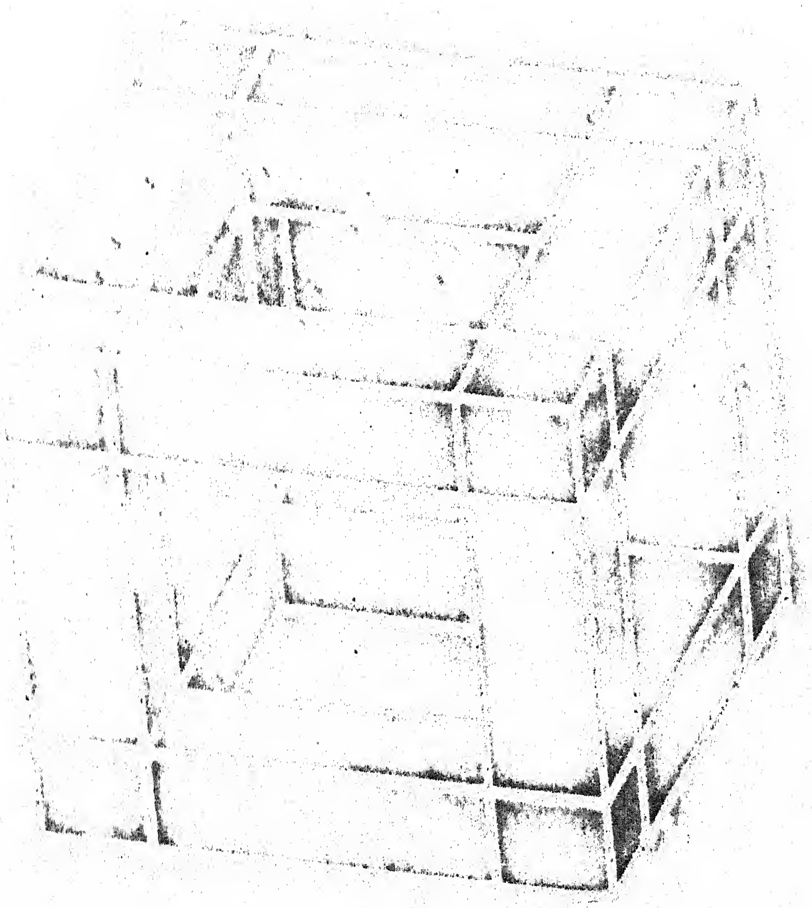
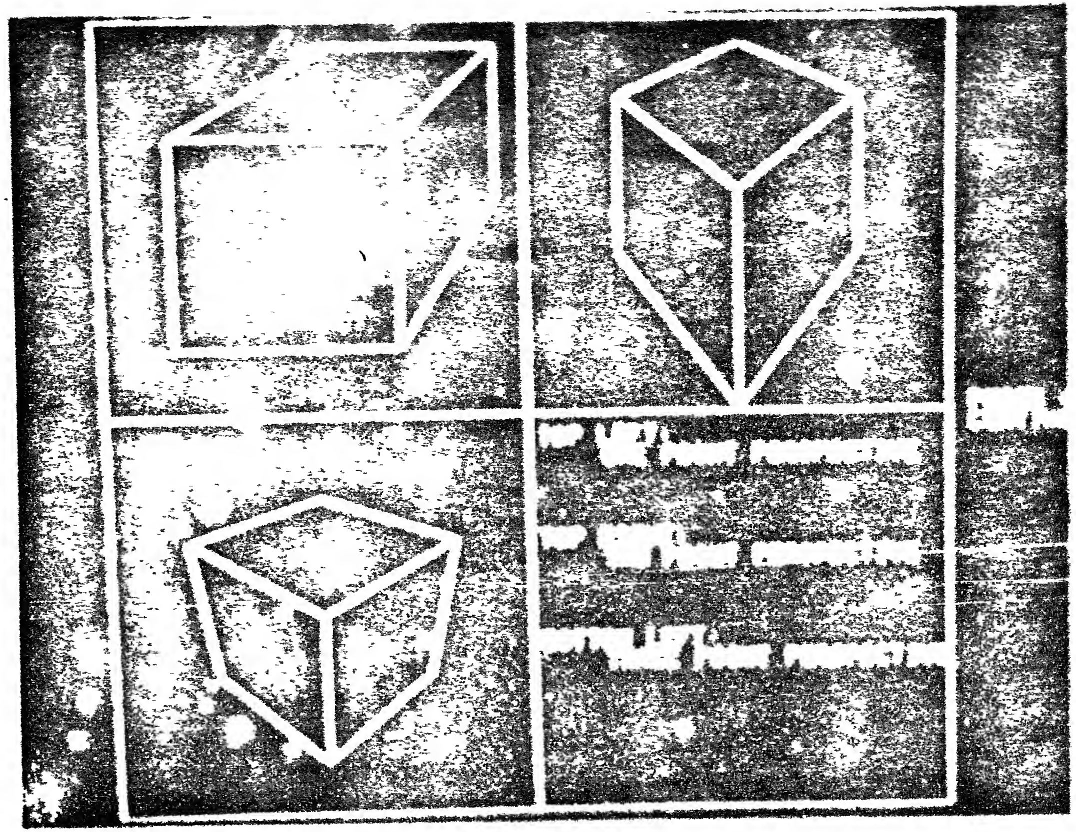


Fig. 4.5: HOLLOW CUBE



. 4.5: PERSPECTIVE PROJECTIONS

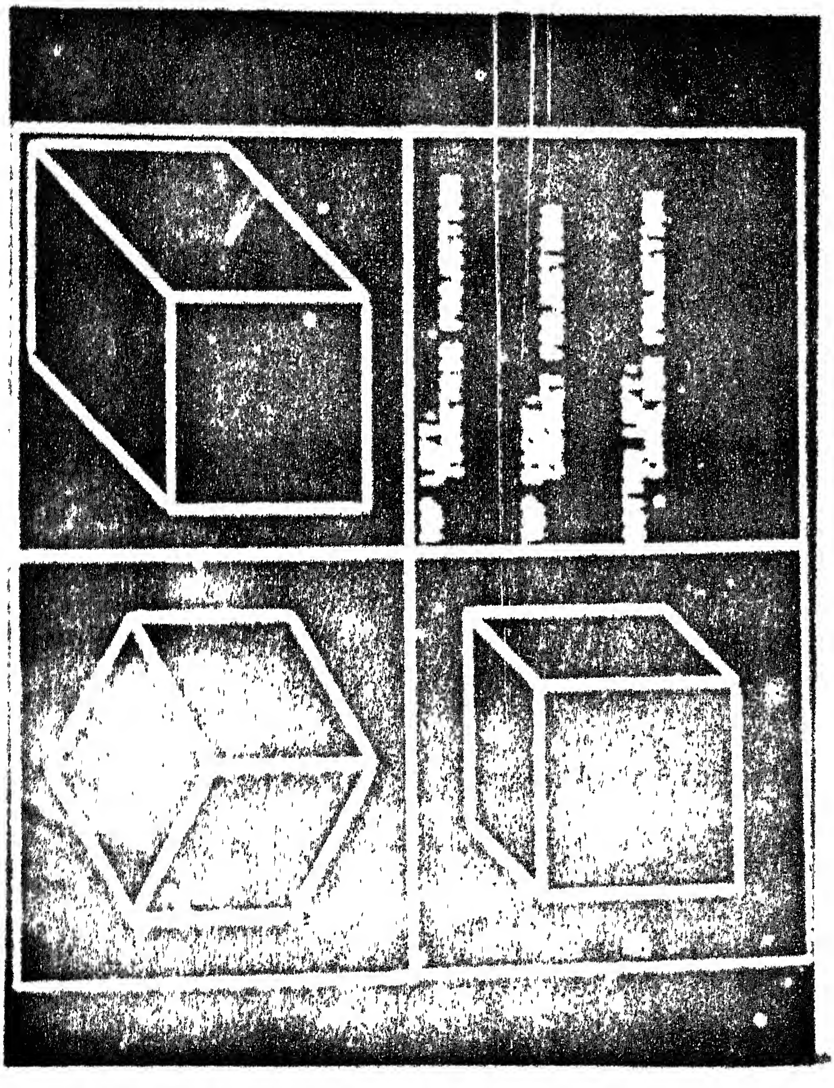


Fig. 4.7: PARALLEL PROJECTION

CHAPTER 5

CONCLUSIONS

5.1 TECHNICAL SUMMARY:

We perceive and process data rapidly and efficiently when they are presented pictorially rather than as text. As such many applications will require the display of three dimensional objects. If the architect would like to see, how the structure will actually look, then a three dimensional model can allow him to view the structure from different points. The aircraft designer may wish to model the behaviour of the craft under three dimensional forces and stresses.

In order to enable the application programmer to display three dimensional objects with ease, a software package consisting of a set of procedures coded in 'SIMULA', a high-level programming language, was developed. The file named 'GRAS' contains these procedures. Facilities provided by this package are (1) Different types of planar geometric projections (2) Instant transformations (3) picture segmentation (4) Hidden line/surface elimination. These facilities are made available through the command set. The package is primarily designed for Tektronix 4006-1

graphics terminal, which is of 'Direct View Storage Tube' type. The graphics package can be modified to other display devices by modifying the 'display code generator' part of the package. Since the package is written in a high-level programming language namely SIMULA except for one procedure, this package can be implemented on any machine that has Simula compiler by rewriting that procedure alone.

5.2 FURTHER SCOPE:

The facilities provided will be adequate for a majority of graphics applications. Simple and straight-forward algorithms have been used in the design of the package. Input functions are not included in the package as we are using Direct View Storage Tube' type display. The package can be extended to include input functions to cater for refresh type display. The algorithm used for hidden line elimination is not efficient. Many hidden line algorithms consume a lot of computer time and computer storage. If better algorithm is invented in due course, it can be incorporated. For display file, the data structure 'array' is being used. A free storage allocation system can be employed to supply blocks of free memory and to receive blocks that are vacated, to manage the storage efficiently.

APPENDIX A

THREE DIMENSIONAL TRANSFORMATIONS

1. Translation:

Let x,y,z be the coordinates of a point in a 3-dimensional coordinate system. When this point is translated by Tx, Ty, Tz in x,y,z directions, its coordinates get altered. Let us denote the altered coordinates of the point by x', y', z'. The values of x',y',z' are given by

$$x' = x + Tx \qquad (1)$$

$$y' = y + Ty \qquad (2)$$

$$z' = z + Ty \qquad (3)$$

Equations (1),(2) and (3) when written in matrix form will be of the form

$$[x' \; y' \; z'] = [x+Tx \quad y+Ty \quad z+Tz]$$

$$= [x \; y \; z] + [Tx \; Ty \; Tz] \qquad (4)$$

By denoting the row vectors [x y z], [x' y' z'] and [Tx Ty Tz] by P, P' and T respectively, we can rewrite equation (4) concisely

$$P' = P+T \qquad (5)$$

By translating all the points on the object, the object can be translated.

2. **Scaling:**

Let x,y,z be the coordinates of a point in a 3-dimensional coordinate system. When this point is stretched by Sx along x-axis, by Sy along y-axis and by Sz along z-axis its coordinates get altered. Let us denote the altered coordinates of the point by x',y',z'. The values of x',y', and z' are given by

$$x' = (x) * (Sx) \tag{6}$$

$$y' = (y) * (Sy) \tag{7}$$

$$z' = (z) * (Sz) \tag{8}$$

Equation (6) and (7) and (8) when in written matrix form, will be of the form

$$[x' \ y' \ z'] = [x*Sx \quad y*Sy. \quad z*Sz]$$

$$= [x \ y \ z]* \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & Sz \end{bmatrix} \tag{9}$$

By defining P, P' and S as shown below

$$P = [x \ y \ z]$$

$$P' = [x' \ y' \ z']$$

$$S = \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & Sz \end{bmatrix}$$

We can rewrite equation (9) concisely as

$$P' = P*S \tag{10}$$

Rotation:

Let a point (x,y,z) when rotated through an angle θ anticlock wise about z axis be transformed into a new point (x',y',z'). The values of x',y' and z' are given by

$$x' = x \cos\theta - y \sin\theta \tag{11}$$

$$y' = y \cos\theta + x \sin\theta \tag{12}$$

$$z' = z \tag{13}$$

Equations (11), (12) and (13) when written in matrix form, will be as follows

$$[x' \; y' \; z'] = [x \cos\theta - y \sin\theta \quad y \cos\theta - x \sin\theta \quad z]$$

$$= [\, x \; y \; z \,] * \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

By defining P, P' and Rz as shown below

$$P = [x \; y \; z]$$

$$P' = [x' \; y' \; z']$$

$$Rz(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

~~We can rewrite equation (14)~~ concisely as

$$P' = P * Rz(\theta) \qquad (15)$$

Similarly ~~we~~ can write for rotations about x and y axes as

$$P' = P * Rx(\theta)$$

$$P' = P * Ry(\theta)$$

where,

$$Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}$$

$$Ry(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}$$

The matrix representation for translation, scaling and rotation are shown below.

$$P' = P+T$$

$$P' = P*S$$

$$P' = P*R$$

From the above it can be noticed that the translation is treated differently from the other two.

By transforming the points into homogeneous coordinate system, all the three transformations can be made as multiplications.

In homogeneous coordinates, a point P(x,y,z) is represented as (w.x, w.y, w.z, w) where $w \neq 0$. Given a homogeneous coordinate representation for a point P(X, Y,Z,W), we can find 3d cartesian coordinate representation for the point as $x = \frac{X}{W}$; $y = \frac{Y}{W}$; $z = \frac{Z}{W}$.
If w = 1, 3d-cartesion coordinates can be obtained without division.

Transformations for translation, scaling and rotation, when co-ordinates are represented in homogeneous coordinate systems, are derived and given below.

Translation:

$$[x' \ y' \ z' \ 1] = [x+dx \quad y+dy \quad z+dz \quad 1]$$

$$= [x \ y \ z \ 1] * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ dx & dy & dz & 1 \end{bmatrix}$$

Scaling:

$$[x' \ y' \ z' \ 1] = [x*Sx \quad y*Sy \quad z*Sz \quad 1]$$

$$= [x \ y \ z \ 1] * \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about z-axis:

$$[x' \; y' \; z' \; 1] = [x\cos\theta - y\sin\theta \quad y\cos\theta + x\sin\theta \quad z \quad 1]$$

$$= [x \; y \; z \; 1] * \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly rotations about x and y axes are given below

$$Rx(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Ry(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

APPENDIX B

PLANAR GEOMETRIC PROJECTIONS

To produce a two dimensional view of an object, each point on the object must be mapped to some point on the view surface. The standard method used for generating a two dimensional view of a three dimensional object is by projecting straight lines that are emanating from a fixed point and passing through each point of the object, onto a plane positione in their way, and then finding the image by the intersections of these lines with the plane. The fixed point is known as 'center of projection.' The straight lines are known as 'projectors'. The plane is known as 'projection plane' or 'view plane'. This type of projection is known as 'planar geometric projection' because the projection is onto a plane, rather than onto a some curved surface and projectors are straight, rather than curved. Non-planar and non-geometric projections are extensively used in cartography.

Planar geometric projections can be divided into two classes

1. Perspective projections
2. Parallel projections

If the distance of the object from the 'center of projection' is finite then the projection obtained is 'perspective

projection'. If this distance is infinite, then all the projectors become parallel to each other and the projection obtained is known as 'Parallel Projection'.

PERSPECTIVE PROJECTION:

A perspective projection gives a natural appearance (realism) of an object as seen by the eye, but it is more difficult to construct manually compared to parallel views. However, with computer graphics frame work, once the object has been completely defined, any kind of view can be obtained with equal ease. In this projection objects of equal size appear smaller as their distance from the 'centre of projection' increases, and become larger as this distance decreases. This property is known as 'perspective fore-shortening'.

FEATURES:

1. Only lines parallel to view-plane remain parallel.
2. Angles are preserved on those faces of the object which are parallel to projection plane.
3. Because of perspective foreshortening equal line segments parallel to any particular axis will be foreshortened to different scales and therefore drawings using this projection cannot be used for obtaining dimensions of objects.

Perspective projection of any set of parallel lines of the object that are not parallel to the view-plane, converge to a single point in the drawing. This point is called 'Vanishing point'. The vanishing point is the point, where a line through the 'Center of projection' parallel to the set of the parallel lines, intersects the view-plane. If the set of the parallel lines is parallel to one of the principal axes, this point is called 'Principal vanishing point'. The number of principal vanishing points is equal to the number of principal axes, cut by the view-plane. Perspective projections are classified into three types depending upon the number of principal vanishing points.

Two point perspective projection with view-plane vertical is widely used, whenever a realistic appearance of an object is desired, such as in advertisement drawing and presentation drawings of architectural, engineering, industrial, design. Three point perspectives are not used nearly so much, in part because they are hard to construct and in part because they do not add much realism beyond that afforded by the two point perspective.

PARALLEL PROJECTION:

Parallel projection is less realistic view because it lacks the property of perspective foreshortening. Although

there can be different constant foreshortenings along each axis, exact dimensions of the object can be obtained by multiplying the measurements in the drawing with appropriate scale factor. Parallel lines in the object do remain parallel in the projection. As with perspective, angles are preserved on those faces of the object which are parallel to the projection plane.

Based on the angle between the direction of projectors and the view-plane, parallel projections are categorised into two types.

1. Orthographic parallel projections and
2. Oblique parallel projection

When projectors are perpendicular to the view-plane, then the projection is orthographic; otherwise it is oblique.

ORTHOGRAPHIC PARALLEL PROJECTION:

Depending on the orientation of the view-plane, these are classified into either 'Multi view orthographic projections' or 'Axonometric projections'.

'Multi view orthographic projections' contain two or more orthographic projections planes which are parallel to principal planes. A way of creating a Multi-view orthographic projection is to surround the object with six view-plane which

form a rectangular box around the object and arranging the six orthographic projections in a particular manner.

Six views are rarely used. Mostly front, top, and side projections are used. Multi-view orthographic projections are presented with hidden lines either entirely omitted or drawn using dotted lines. Inclusion of hidden lines enhances the information content of each view. Consequently, one can do with less number of views. If the object is complicated, hidden lines would be of little help as they would clutter up the view of the object. In such circumstances, one should make use of additional views with hidden lines removed.

For objects with complicated internal structure, multi-view orthographic projections prove to be far from adequate. Internal details are best depicted by what are known as 'Sectional views'. A sectional view of an object is obtained by cutting the object with a plane, removing one part of the object, and projecting the remaining part orthogonally onto the cutting plane.

Multi-view orthographic projections are used in engineering drawings to depict machine parts, assemblies,, and buildings because these allow true measurement of distances and angles from these drawings. But since each

depicts only one face of of an object, 3-D nature of the object can be hard to visualise, even if several projections of the same object are studied simultaneously.

Axonometric projection uses projection plane that intersects all three principal axes and therefore show several faces of the object in one view, just like perspective projection. But the foreshortening along any particular axis is constant. Since none of the faces is represented in its true form, use of these projections is limited to rectangular objects. Objects with, circular or curved shaped cross sections, are not represented effectively by this class of projections. Axonometric projections are used in catalogue illustrations, patent office records, piping diagrams, furniture design, machine design, structural design.

Based on the angles, the view-plane normal makes with each principal axis, the axonometric projections are further classified into

1. Isometric projections,
2. Dimetric projections,
3. Trimetric projections.

In isometric projection, the view-plane normal makes equal angles with each principal axis. The isometric

projection has the useful property that, all the three principal axes are equally foreshortened, allowing measurements along the axes to be made with the same scale. (Hence the name: for equal, metric for measure). In addition, the principal axes project so,that they make equal angles with each other.

In dimetric projections, only two angles out of three, are equal. In trimetric projections, all three angles are different.

OBLIQUE PROJECTIONS:

An oblique projection uses view-plane that is parallel to one of the principal planes and all projectors are inclined with the view-plane at an angle other than 90 degrees. This projection has almost the same range of applications as an axonometric representation. However, while axonometric projections are primarily used for rectangular objects, oblique projections are well suited for objects with cylindrical shapes. Oblique projections combine properties of multi-view orthographic projections (which allow measurement of angles and distances) with those of axonometric projections (exhibition of all three principal faces in one view, and measurement of distances to the scale).

There are two important oblique projections. (1) cavalier projection, and (2) cabinet projection. In case of cavalier projection, the projectors make an angle of 45 degrees with the view-plane. As a result, the length of the projection of a line that is perpendicular to the view-plane is same as the line itself, that is there is no foreshortening. In case of cabinet projections, projectors make an angle of arccot (1/2), with the view-plane, so that lines which are perpendicular to view-plane project at 1/2 of their actual length. In the drawings obtained using oblique projections, angle between the projection of the lines that is 'Perpendicular to the view-plane' and horizontal line is usually 35 or 45 degrees.

REFERENCES

1. NEWMAN, W.M. and R.F. SPROULL, 'Principles of Interactive Computer Graphics', 2nd Ed., McGraw-Hill, New York, (1979).

2. Foley, J.D. and A. Vandam; 'Fundamentals of Interactive Computer Graphics', Addison-Wesley , Mass. (1982).

3. ANGEL, I.O., 'A Practical Introduction to Computer Graphics', MacMillan, London (1981).

4. HARRINGTON, S., 'Computer Graphics: A Programming Approach', McGraw-Hill, New York (1983).

5. ROGERS, D.F. and J.A. ADAMS, 'Mathematical Elements for Computer Graphics', McGraw-Hill, New York, (1976).

6. GILOI, W.K., 'Interactive Computer Graphics - Data Structures, Algorithms, Languages', Prentice-Hall, Englewood Cliffs, N.J. (1978).

7. ACM Computing Surveys; vol. 10, No. 4, Dec. 1978.

```
COMMENT! SIMULA specification;
OPTIONS(/E:QUICK,RUBOUT);
PROCEDURE RUBOUT;;

!;! MACRO-10 code

TITLE   RUBOUT              ;!PROCEDURE TO RUBOUT TEKTRONIX SCREEN
ENTRY   RUBOUT              ;!************************************

RUBOUT:MOVEI 1,33
OUTCHR 1
MOVEI 1,14
IONEOU 1
POPJ 17,
END;
```

```
OPTIONS(/c);
EXTERNAL PROCEDURE ROBOUT;
CLASS GRAS;
BEGIN
    INTEGER ARRAY START,SIZE[0:10];
    BOOLEAN ARRAY VISIBILITY,PAINTED[0:10];
    INTEGER BFREE,CFREE,DFREE,DF_FREE,FREE,COUNT_OUT;
    INTEGER ARRAY OPB[1:32],OPC,OPD[1:1000],DF_OP[1:4000];
    REAL ARRAY VPTM,ITM,CTM[1:4,1:3];
    REAL ARRAY XB,YB,ZB[1:32];
    REAL ARRAY XC,YC,ZC[1:1000];
    REAL ARRAY XD,YD,ZD[1:1000];
    REAL ARRAY DF_X,DF_Y[1:4000];
    REAL WXL,WXH,WYL,WYH;
    REAL VXL,VXH,VYL,VYH;
    REAL XLM,XHM,YLM,YHM;
    REAL WVSX,WVSY,WVTX,WVTY;
    REAL LAST_FCX,LAST_FCY,LAST_FCZ;
    REAL LAST_RRCX,LAST_RRCY,LAST_RRCZ;
    REAL LAST_BCX,LAST_BCY,LAST_BCZ;
    REAL LAST_TCX,LAST_TCY,LAST_TCZ;
    REAL LAST_RTCX,LAST_RTCY,LAST_RTCZ;
    REAL LAST_LCX,LAST_LCY,LAST_LCZ;
    BOOLEAN HIDDEN_FLAG,WIRE_FRAME,CLOSED_SURFACE,ERASE;
    BOOLEAN FRONT_FLAG,REAR_FLAG,PFLAG,PERSPECTIVE_FLAG;
    INTEGER ARRAY
    TO_BE_TESTED[1:100],START_INDEX,BACK_TOTAL[1:1000];
    INTEGER ARRAY INFO,LINK[1:2000];
    INTEGER TEST_TOTAL,CURRENTLY_OPEN,MAX_LIMIT;
    CHARACTER CH;
    REAL FZ_CLIP,RZ_CLIP;
    REAL XS,YS,XF,YF,XP,YP,XO,YO;
    REAL SVX,SVY,SVZ;
    REAL SVTX,SVTY;
    REAL VPTX,VPTY, VPTZ;
    REAL CX,CY,CZ;
    REAL EPSILON;
    PROCEDURE SETUP_VIEWPLANE(RX,RY,RZ,NX,NY,NZ,D,PX,PY,PZ);
    REAL RX,RY,RZ,
    NX,NY,NZ,
    PX,PY;PZ,
    D;
    !-----------------------------------------------

    "rx,ry,rz": coordinates of reference point
    in the world coordinate units.
    "nx,ny,nz": coordinates of any one point on
    view plane normal measured relative to the
    reference point in the world coordinate
    units.
    "px,py,pz":coordinates of the end point of
    a vector positioned at reference point
    measured relative to the reference point
    in world coordinate units.
    d     :distance of view plane from reference
    point.
    -----------------------------------------------;
    BEGIN
        REAL TEMP,PRX,PRY,PRD;
        TEMP:=SQRT(NX^2+NY^2+NZ^2);
        NX:=NX/TEMP;
        NY:=NY/TEMP;
        NZ:=NZ/TEMP;
        SET_VPTM_TO_IDENTITY;
        TRANSLATE_3(-(RX+(D*NX)),-(RY+(D*NY)),-(RZ+(D*NZ)));
        TEMP:=SQRT(NY^2+NZ^2);
        ROTATE_3_X(NY/TEMP,NZ/TEMP);
        ROTATE_3_Y(-NX,TEMP);
        PRX:=(PX*VPTM[1,1])+(PY*VPTM[2,1])+(PZ*VPTM[3,1]);
        PRY:=(PX*VPTM[1,2])+(PY*VPTM[2,2])+(PZ*VPTM[3,2]);
        PRD:=SQRT(PRX^2+PRY^2);
        ROTATE_3_Z(PRX/PRD,PRY/PRD);
        END_XFORM;
    END;
    PROCEDURE SET_PROJECTION_PARAMETERS(VX,VY,VZ,FLAG);
    REAL     VX,VY,VZ;
    BOOLEAN  FLAG;
    !-----------------------------------------------
    in order to have parrallel projection flag shoud be
    false and vx,vy,vz shoud be x,y,z values of viewing
    direction in view plane co-ordinate system.

    in order to have perspective projection flag shoud be
    true and vx,vy,vz shoud be x,y,z co-ordinates of
    view point in view plane co-ordinate system.
    -----------------------------------------------;
    BEGIN
        PERSPECTIVE_FLAG:=FLAG;
        SVX:=VX;
```

```
        SVY:=VY;
        SVZ:=VZ;
        IF PERSPECTIVE_FLAG THEN
        BEGIN
             XLM:=(VX-WXL)/VZ;
             XHM:=(VX-WXH)/VZ;
             YLM:=(VY-WYL)/VZ;
             YHM:=(VY-WYH)/VZ;
        END
        ELSE
        BEGIN
             XLM:=XHM:=VX/VZ;
             YLM:=YHM:=VY/VZ;
        END;
   END;
   PROCEDURE SET_WINDOW(XL,XH,YL,YH);
   REAL XL,XH,YL,YH;
   BEGIN
        WXL:=XL;
        WXH:=XH;
        WYL:=YL;
        WYH:=YH;
        IF PERSPECTIVE_FLAG THEN
        BEGIN
             XLM:=(SVX-WXL)/SVZ;
             XHM:=(SVX-WXH)/SVZ;
             YLM:=(SVY-WYL)/SVZ;
             YHM:=(SVY-WYH)/SVZ;
        END;
        NEW_VIEWING_TRANSFORMATION;
   END;
   PROCEDURE SET_VIEWPORT(XL,XH,YL,YH);INTEGER XL,XH,YL,YH;
   BEGIN
        VXL:=XL;
        VXH:=XH;
        VYL:=YL;
        VYH:=YH;
        DRAW_VIEWPORT;
        NEW_VIEWING_TRANSFORMATION;
   END;
   PROCEDURE DRAW_VIEWPORT;
   BEGIN
        OUTCHAR(CHAR(29));
        TRANSMIT_COORDINATES(VXL,VYL);
        TRANSMIT_COORDINATES(VXH,VYL);
        TRANSMIT_COORDINATES(VXH,VYH);
        TRANSMIT_COORDINATES(VXL,VYH);
        TRANSMIT_COORDINATES(VXL,VYL);
        BREAKUPIMAGE;
   END;
   PROCEDURE SET_DEPTH_FLAGS(FFLAG,RFLAG);
   BOOLEAN FFLAG,RFLAG;
   BEGIN
        FRONT_FLAG:=FFLAG;
        REAR_FLAG:=RFLAG;
   END;
   PROCEDURE SET_VIEW_DEPTH(FPD,BPD);
   REAL FPD,BPD;
   BEGIN
        FZ_CLIP:=-FPD;
        RZ_CLIP:=-BPD;
   END;
   PROCEDURE SET_ITM_TO_IDENTITY;
   BEGIN
        INTEGER I,J;
        FOR I:=1 STEP 1 UNTIL 4 DO
        BEGIN
             FOR J:=1 STEP 1 UNTIL 3 DO ITM[I,J]:=0;
             IF I \= 4 THEN ITM[I,I]:=1;
        END;
   END;
   PROCEDURE TRANSLATE_I(TX,TY,TZ);
   REAL TX,TY,TZ;
   BEGIN
        ITM[4,1]:=ITM[4,1]+TX;
        ITM[4,2]:=ITM[4,2]+TY;
        ITM[4,3]:=ITM[4,3]+TZ;
   END;
   PROCEDURE ROTATE_I(AXIS,THEATA);
   INTEGER AXIS;
   REAL THEATA;
   BEGIN
        INTEGER I1,I2,K;
        REAL ARRAY TEMP[1:3];
        REAL ARRAY TM[1:3,1:3];
        REAL CT,ST;
        THEATA:=THEATA*3.1415926535/180;
        ST:=SIN(THEATA);
        CT:=COS(THEATA);
```

```
     FOR I1:= 1 STEP 1 UNTIL 3 DO
     FOR I2:= 1 STEP 1 UNTIL 3 DO
     TM[I1,I2]:=0;
     TM[AXIS,AXIS]:=1;
     I1:=MOD(AXIS,3)+1;
     I2:=MOD(I1,3)+1;
     TM[I1,I1]:=CT;
     TM[I2,I2]:=CT;
     TM[I1,I2]:=ST;
     TM[I2,I1]:=-ST;
     FOR I1:=1 STEP 1 UNTIL 4 DO
     BEGIN
          FOR I2:=1 STEP 1 UNTIL 3 DO
          BEGIN
               TEMP[I2]:=0;
               FOR K:=1 STEP 1 UNTIL 3 DO
               TEMP[I2]:=TEMP[I2]+ITM[I1,K]*TM[K,I2];
          END;
          FOR K:=1 STEP 1 UNTIL 3 DO ITM[I1,K]:=TEMP[K];
     END;
END;
PROCEDURE SCALE_I(SX,SY,SZ);
REAL SX,SY,SZ;
BEGIN
     INTEGER I;
     FOR I:=1 STEP 1 UNTIL 4 DO
     BEGIN
          ITM[I,1]:=ITM[I,1]*SX;
          ITM[I,2]:=ITM[I,2]*SY;
          ITM[I,3]:=ITM[I,3]*SZ;
     END;
END;
PROCEDURE BEGIN_XFORM;
BEGIN
     INTEGER I,J,K;
     FOR I:=1 STEP 1 UNTIL 4 DO
     FOR J:=1 STEP 1 UNTIL 3 DO
     BEGIN
          CTM[I,J]:=0;
          FOR K:=1 STEP 1 UNTIL 3 DO
          CTM[I,J]:=CTM[I,J]+ITM[I,K]*VPTM[K,J];
     END;
     FOR J:=1 STEP 1 UNTIL 3 DO CTM[4,J]:=CTM[4,J]+VPTM[4,J];
     SET_ITM_TO_IDENTITY;
END;
PROCEDURE END_XFORM;
BEGIN
     INTEGER I,J;
     FOR I:=1 STEP 1 UNTIL 4 DO
     FOR J:=1 STEP 1 UNTIL 3 DO
     CTM[I,J]:=VPTM[I,J];
END;
PROCEDURE CREATE_SEGMENT(SEG_NO);
INTEGER SEG_NO;
BEGIN
     IF CURRENTLY_OPEN > 0 THEN CLOSE_SEGMENT;
     IF SEG_NO < 1 OR
     SEG_NO > MAX_LIMIT THEN
     BEGIN
          OUTTEXT("OPENING SEGMENT WITH INVALID SEGMENT NUMBER");
          GOTO LAST;
     END;
     IF SIZE[SEG_NO] > 0 THEN
     BEGIN
          OUTTEXT("OPENING SEGMENT THAT EXISTS ALREADY");
          GOTO LAST;
     END;
     START[SEG_NO]:=DF_FREE;
     SIZE[SEG_NO]:=0;
     VISIBILITY[SEG_NO]:=FALSE;
     CURRENTLY_OPEN:=SEG_NO;
     LAST:
END;
PROCEDURE CLOSE_SEGMENT;
BEGIN
     IF CURRENTLY_OPEN = 0 THEN GOTO LAST;
     CURRENTLY_OPEN:=0;
     DELETE_SEGMENT(0);
     START[0]:=DF_FREE;
     SIZE[0]:=0;
     LAST:
END;
PROCEDURE DELETE_SEGMENT(SEG_NO);
INTEGER SEG_NO;
BEGIN
     INTEGER I,GET,PUT,LENGTH,OP;
     REAL X,Y;
     IF SEG_NO < 0 OR
     SEG_NO > MAX_LIMIT THEN
```

```
    BEGIN
        OUTTEXT("DELETING SEGMENT WITH INVALID SEGMENT NUMBER");
        GOTO LAST;
    END;
    IF CURRENTLY_OPEN \= 0 THEN CLOSE_SEGMENT;
    IF SIZE[SEG_NO] = 0 THEN GOTO LAST;
    PUT:=START[SEG_NO];
    LENGTH:=SIZE[SEG_NO];
    GET:=PUT+LENGTH;
    WHILE GET < DF_FREE DO
    BEGIN
        GET_INSTRUCTION(GET,OP,X,Y);
        PUT_INSTRUCTION(PUT,OP,X,Y);
        GET:=GET+1;
        PUT:=PUT+1;
    END;
    DF_FREE:=PUT;
    FOR I:=0 STEP 1 UNTIL MAX_LIMIT DO

    IF START[I] > START[SEG_NO] THEN START[I]:=
    START[I]-SIZE[SEG_NO];
    SIZE[SEG_NO]:=0;
    IF VISIBILITY[SEG_NO]  THEN
    BEGIN
        ERASE:=TRUE;
        VISIBILITY[SEG_NO]:=FALSE;
    END;
    LAST:
END;
PROCEDURE PUT_INSTRUCTION(POS,OP,X,Y);
INTEGER POS,OP;
REAL X,Y;
BEGIN
    DF_OP(POS):=OP;
    DF_X(POS):=X;
    DF_Y(POS):=Y;
END;
PROCEDURE GET_INSTRUCTION(POS,OP,X,Y);
NAME OP,X,Y;
INTEGER POS,OP;
REAL X,Y;
BEGIN
    OP:=DF_OP(POS);
    X:=DF_X(POS);
    Y:=DF_Y(POS);
END;
PROCEDURE DELETE_ALL_SEGMENTS;
BEGIN
    INTEGER I;
    FOR I:=0 STEP 1 UNTIL MAX_LIMIT DO
    BEGIN
        START[I]:=1;
        SIZE[I]:=0;
    END;
    CURRENTLY_OPEN:=0;
    DF_FREE:=1;
    ERASE:=TRUE;
END;
PROCEDURE POST_SEGMENT(SEG_NO);
INTEGER SEG_NO;
BEGIN
    IF SEG_NO < 0 OR
    SEG_NO > MAX_LIMIT THEN
    GOTO LAST;
    IF CURRENTLY_OPEN > 0 THEN CLOSE_SEGMENT;
    VISIBILITY[SEG_NO]:=TRUE;
    LAST:
END;
PROCEDURE UNPOST_SEGMENT(SEG_NO);
INTEGER SEG_NO;
BEGIN
    IF SEG_NO < 0 OR
    SEG_NO > MAX_LIMIT THEN
    GOTO LAST;
    IF CURRENTLY_OPEN > 0 THEN CLOSE_SEGMENT;
    IF SIZE[SEG_NO] \= 0 AND  VISIBILITY[SEG_NO]  THEN
    ERASE:=TRUE;
    VISIBILITY[SEG_NO]:=FALSE;
    LAST:
END;
PROCEDURE UPDATE_DISPLAY;
BEGIN
    INTEGER I;
    IF ERASE THEN
    BEGIN
        CLEAR_SCREEN;
        FOR I:=0 STEP 1 UNTIL MAX_LIMIT DO
        IF SIZE[I] \= 0 AND VISIBILITY[I] THEN
        BEGIN
```

```
                INTERPRET(START[I],SIZE[I]);
                PAINTED[I]:=TRUE;
            END
            ELSE
            BEGIN
                PAINTED[I]:=FALSE;
            END;
            ERASE:=FALSE;
        END
        ELSE
        BEGIN
            FOR I:=0 STEP 1 UNTIL MAX_LIMIT DO
            IF SIZE[I] \= 0 AND VISIBILITY[I] AND NOT PAINTED[I] THEN
            BEGIN
                INTERPRET(START[I],SIZE[I]);
                PAINTED[I]:=TRUE;
            END;
        END;
END;
PROCEDURE INTERPRET(FIRST,WIDTH);
INTEGER FIRST,WIDTH;
BEGIN
    INTEGER I,LAST,OP;
    REAL X,Y;
    LAST:=FIRST+WIDTH-1;
    FOR I:= FIRST STEP 1 UNTIL LAST DO
    BEGIN
        GET_INSTRUCTION(I,OP,X,Y);
        IF OP=2 THEN SHOW_LINE(SVTX,SVTY,X,Y);
        SVTX:=X;
        SVTY:=Y;
    END;
END;
PROCEDURE A_MOVE_3(X,Y,Z);REAL X,Y,Z;
BEGIN
    CX:=X;
    CY:=Y;
    CZ:=Z;
    DO_CURRENT_TRANSFORMATION(CX,CY,CZ);
    CLIP_LEFT(1,VPTX,VPTY,VPTZ);
END;
PROCEDURE R_MOVE_3(DX,DY,DZ);REAL DX,DY,DZ;
BEGIN
    CX:=CX+DX;
    CY:=CY+DY;
    CZ:=CZ+DZ;
    DO_CURRENT_TRANSFORMATION(CX,CY,CZ);
    CLIP_LEFT(1,VPTX,VPTY,VPTZ);
END;
PROCEDURE A_LINE_3(X,Y,Z);REAL X,Y,Z;
BEGIN
    CX:=X;
    CY:=Y;
    CZ:=Z;
    DO_CURRENT_TRANSFORMATION(CX,CY,CZ);
    CLIP_LEFT(2,VPTX,VPTY,VPTZ);
END;
PROCEDURE R_LINE_3(DX,DY,DZ);REAL DX,DY,DZ;
BEGIN
    CX:=CX+DX;
    CY:=CY+DY;
    CZ:=CZ+DZ;
    DO_CURRENT_TRANSFORMATION(CX,CY,CZ);
    CLIP_LEFT(2,VPTX,VPTY,VPTZ);
END;
PROCEDURE A_POLYGON_3(AX,AY,AZ,N);
REAL ARRAY AX,AY,AZ;
INTEGER N;
BEGIN
    INTEGER I;
    REAL TX,TY,TZ;
    REAL ARRAY BX,BY,BZ[1:4];
    CX:=AX[1];
    CY:=AY[1];
    CZ:=AZ[1];
    FOR I:=1 STEP 1 UNTIL N DO
    BEGIN
        TX:=AX[I];
        TY:=AY[I];
        TZ:=AZ[I];
        DO_CURRENT_TRANSFORMATION(TX,TY,TZ);
        BX[I]:=VPTX;
        BY[I]:=VPTY;
        BZ[I]:=VPTZ;
    END;
    POLYGON(BX,BY,BZ,N);
END;
PROCEDURE R_POLYGON_3(AX,AY,AZ,N);
REAL ARRAY AX,AY,AZ;
```

```
INTEGER N;
BEGIN
    INTEGER I;
    REAL TX,TY,TZ;
    REAL ARRAY BX,BY,BZ[1:4];
    TX:=AX[1]+CX;
    TY:=AY[1]+CY;
    TZ:=AZ[1]+CZ;
    FOR I:=1 STEP 1 UNTIL N DO
    BEGIN
        CX:=AX[I]+CX;
        CY:=AY[I]+CY;
        CZ:=AZ[I]+CZ;
        DO_CURRENT_TRANSFORMATION(CX,CY,CZ);
        BX[I]:=VPTX;
        BY[I]:=VPTY;
        BZ[I]:=VPTZ;

    END;
    CX:=TX;
    CY:=TY;
    CZ:=TZ;
    POLYGON(BX,BY,BZ,N);
END;
PROCEDURE POLYGON(AX,AY,AZ,N);
REAL ARRAY AX,AY,AZ;
INTEGER N;
BEGIN
    INTEGER I;
    PFLAG:=TRUE;
    COUNT_OUT:=0;
    LAST_LCX:=LAST_RTCX:=LAST_BCX:=LAST_TCX:=LAST_RRCX:=LAST_FCX
    :=
    AX[N];
    LAST_LCY:=LAST_RTCY:=LAST_BCY:=LAST_TCY:=LAST_RRCY:=LAST_FCY
    :=
    AY[N];
    LAST_LCZ:=LAST_RTCZ:=LAST_BCZ:=LAST_TCZ:=LAST_RRCZ:=LAST_FCZ
    :=
    AZ[N];
    FOR I:=1 STEP 1 UNTIL N DO
    CLIP_LEFT(2,AX[I],AY[I],AZ[I]);
    IF COUNT_OUT > 0 THEN
    CLIP_LEFT(2,XB[1],YB[1],ZB[1]);
    COUNT_OUT:=COUNT_OUT-1;
    PFLAG:=FALSE;
    IF COUNT_OUT < 3 THEN GO TO LAST;
    IF COUNT_OUT >=32 THEN GO TO LAST;
    IF WIRE_FRAME THEN
    BEGIN
        DO_PROJECTION(COUNT_OUT,XB[COUNT_OUT],YB[COUNT_OUT],
        ZB[COUNT_OUT]);
        FOR I:=1 STEP 1 UNTIL COUNT_OUT DO
        DO_PROJECTION(OPB[I],XB[I],YB[I],ZB[I]);
    END
    ELSE
    BEGIN
        IF CLOSED_SURFACE THEN
        BEGIN
            IF IS_BACK_FACE THEN GO TO LAST;
            IF HIDDEN_FLAG THEN
            BEGIN
                TRANSFER_TO_C(COUNT_OUT,XB[COUNT_OUT].YB[COUNT_OUT],
                ZB[COUNT_OUT]);
                FOR I:=1 STEP 1 UNTIL COUNT_OUT DO
                TRANSFER_TO_C(OPB[I],XB[I],YB[I],ZB[I]);
            END
            ELSE
            BEGIN
                DO_PROJECTION(COUNT_OUT,XB[COUNT_OUT].YB[COUNT_OUT],
                ZB[COUNT_OUT]);
                FOR I:=1 STEP 1 UNTIL COUNT_OUT DO
                DO_PROJECTION(OPB[I],XB[I],YB[I],ZB[I]);
            END;
        END
        ELSE
        BEGIN
            TRANSFER_TO_C(COUNT_OUT,XB[COUNT_OUT],YB[COUNT_OUT],
            ZB[COUNT_OUT]);
            FOR I:=1 STEP 1 UNTIL COUNT_OUT DO
            TRANSFER_TO_C(OPB[I],XB[I],YB[I],ZB[I]);
        END;
    END;
    LAST:
END;
PROCEDURE CIRCLE(R,TX,TY,TZ);
REAL R,TX,TY,TZ;
BEGIN
    REAL CT,ST,X,Y,TRX,TRY;
```

```
     INTEGER I;
     CT:=COS(3.141592655З/180);
     ST:=SIN(3.1415926553/180);
     X:=R;
     Y:=0;
     A_MOVE_3(X+TX,TY,TZ);
     FOR I:=1 STEP 1 UNTIL 360 DO
     BEGIN
          TRX:=X*CT-Y*ST;
          TRY:=Y*CT+X*ST;
          X:=TRX;
          Y:=TRY;
          A_LINE_3(X+TX,Y+TY,TZ);
     END;
END;
PROCEDURE SET_VPTM_TO_IDENTITY;
BEGIN
     INTEGER I,J;
     FOR I:=1 STEP 1 UNTIL 4 DO
     BEGIN
          FOR J:=1 STEP 1 UNTIL 3 DO VPTM(I,J):=0;
          IF I NE 4 THEN VPTM(I,I):=1;
     END;
END;
PROCEDURE TRANSLATE_3(X,Y,Z);REAL X,Y,Z;
BEGIN
     VPTM[4,1]:=VPTM[4,1]+X;
     VPTM[4,2]:=VPTM[4,2]+Y;
     VPTM[4,3]:=VPTM[4,3]+Z;
END;
PROCEDURE ROTATE_3_X(SINA,COSA);REAL SINA,COSA;
BEGIN
     INTEGER I;
     REAL TEMP;
     FOR I:=1 STEP 1 UNTIL 4 DO
     BEGIN
          TEMP:=VPTM[I,2]*COSA-VPTM[I,3]*SINA;
          VPTM[I,3]:=VPTM[I,2]*SINA+VPTM[I,3]*COSA;
          VPTM[I,2]:=TEMP;
     END;
END;
PROCEDURE ROTATE_3_Y(SINA,COSA);REAL SINA,COSA;
BEGIN
     INTEGER I;
     REAL TEMP;
     FOR I:=1 STEP 1 UNTIL 4 DO
     BEGIN
          TEMP:=VPTM[I,1]*COSA+VPTM[I,3]*SINA;
          VPTM[I,3]:=-VPTM[I,1]*SINA+VPTM[I,3]*COSA;
          VPTM[I,1]:=TEMP;
     END;
END;
PROCEDURE ROTATE_3_Z(SINA,COSA);REAL SINA,COSA;
BEGIN
     INTEGER I;
     REAL TEMP;
     FOR I:=1 STEP 1 UNTIL 4 DO
     BEGIN
          TEMP:=VPTM[I,1]*COSA-VPTM[I,2]*SINA;
          VPTM[I,2]:=VPTM[I,1]*SINA+VPTM[I,2]*COSA;
          VPTM[I,1]:=TEMP;
     END;
END;
PROCEDURE DO_CURRENT_TRANSFORMATON(X,Y,Z);REAL X,Y,Z;
BEGIN
     VPTX:=CTM[1,1]*X+CTM[2,1]*Y+CTM[3,1]*Z+CTM[4,1];
     VPTY:=CTM[1,2]*X+CTM[2,2]*Y+CTM[3,2]*Z+CTM[4,2];
     VPTZ:=CTM[1,3]*X+CTM[2,3]*Y+CTM[3,3]*Z+CTM[4,3];
END;
PROCEDURE CLIP_LEFT(OP,X,Y,Z);INTEGER OP;REAL X,Y,Z;
BEGIN
     REAL NEW_TEST_X,OLD_TEST_X,X_CLIP,Y_CLIP,Z_CLIP;
     NEW_TEST_X:=WXL+XLM*Z;
     OLD_TEST_X:=WXL+XLM*LAST_LCZ;
     IF (X >= NEW_TEST_X AND LAST_LCX < OLD_TEST_X) OR
     (X <= NEW_TEST_X AND LAST_LCX > OLD_TEST_X) THEN
     BEGIN
          Z_CLIP:=((LAST_LCZ)*(X-LAST_LCX)+(Z-LAST_LCZ)*(WXL-LAST_LCX
          ))/((X-LAST_LCX)-(Z-LAST_LCZ)*(XLM));
          X_CLIP:=WXL+XLM*Z_CLIP;
          Y_CLIP:=GET_Y_CLIP(LAST_LCX,LAST_LCY,LAST_LCZ,X,Y,Z,Z_CLIP,
          X_CLIP);
          IF LAST_LCX < OLD_TEST_X THEN
          CLIP_RIGHT(1,X_CLIP,Y_CLIP,Z_CLIP) ELSE
          CLIP_RIGHT(OP,X_CLIP,Y_CLIP,Z_CLIP);
     END;
     LAST_LCX:=X;
     LAST_LCY:=Y;
     LAST_LCZ:=Z;
```

```
    IF X >= NEW_TEST_X THEN CLIP_RIGHT(OP,X,Y,Z);
END;
PROCEDURE CLIP_RIGHT(OP,X,Y,Z);INTEGER OP;REAL X,Y,Z;
BEGIN
    REAL NEW_TEST_X,OLD_TEST_X,X_CLIP,Y_CLIP,Z_CLIP;
    NEW_TEST_X:=WXH+XHM*Z;
    OLD_TEST_X:=WXH+XHM*LAST_RTCZ;
    IF (X <= NEW_TEST_X AND LAST_RTCX > OLD_TEST_X) OR
    (X >= NEW_TEST_X AND LAST_RTCX < OLD_TEST_X) THEN
    BEGIN
        Z_CLIP:=((LAST_RTCZ)*(X-LAST_RTCX)+(Z-LAST_RTCZ)*(WXH-
        LAST_RTCX))/((X-LAST_RTCX)-(Z-LAST_RTCZ)*(XHM));
        X_CLIP:=WXH+XHM*Z_CLIP;
        Y_CLIP:=GET_Y_CLIP(LAST_RTCX,LAST_RTCY,LAST_RTCZ,X,Y,Z,
        Z_CLIP,X_CLIP);
        IF LAST_RTCX > OLD_TEST_X THEN
        CLIP_BOTTOM(1,X_CLIP,Y_CLIP,Z_CLIP) ELSE
        CLIP_BOTTOM(OP,X_CLIP,Y_CLIP,Z_CLIP);
    END;
    LAST_RTCX:=X;
    LAST_RTCY:=Y;
    LAST_RTCZ:=Z;
    IF X <= NEW_TEST_X THEN CLIP_BOTTOM(OP,X,Y,Z);
END;
REAL PROCEDURE GET_Y_CLIP(X1,Y1,Z1,X2,Y2,Z2,Z_CLIP,X_CLIP);
REAL X1,Y1,Z1,X2,Y2,Z2,Z_CLIP,X_CLIP;
BEGIN
    IF ABS(Z2-Z1) > EPSILON THEN
    GET_Y_CLIP:=Y1+((Y2-Y1)*(Z_CLIP-Z1)/(Z2-Z1))
    ELSE IF ABS(X2-X1) > EPSILON THEN
    GET_Y_CLIP:=Y1+((Y2-Y1)*(X_CLIP-X1)/(X2-X1))
    ELSE GET_Y_CLIP:=Y1;
END;
PROCEDURE CLIP_BOTTOM(OP,X,Y,Z);INTEGER OP;REAL X,Y,Z;
BEGIN
    REAL NEW_TEST_Y,OLD_TEST_Y,X_CLIP,Y_CLIP,Z_CLIP;
    NEW_TEST_Y:=WYL+YLM*Z;
    OLD_TEST_Y:=WYL+YLM*LAST_BCZ;
    IF (Y >= NEW_TEST_Y AND LAST_BCY < OLD_TEST_Y) OR
    (Y <=NEW_TEST_Y AND LAST_BCY > OLD_TEST_Y) THEN
    BEGIN
        Z_CLIP:=(LAST_BCZ*(Y-LAST_BCY)+(Z-LAST_BCZ)*(WYL-LAST_BCY))
        /((Y-LAST_BCY)-(Z-LAST_BCZ)*(YLM));
        Y_CLIP:=WYL+YLM*Z_CLIP;
        X_CLIP:=GET_X_CLIP(LAST_BCX,LAST_BCY,LAST_BCZ,X,Y,Z,Z_CLIP,
        Y_CLIP);
        IF LAST_BCY < OLD_TEST_Y THEN
        CLIP_TOP(1,X_CLIP,Y_CLIP,Z_CLIP)ELSE
        CLIP_TOP(OP,X_CLIP,Y_CLIP,Z_CLIP);
    END;
    LAST_BCX:=X;
    LAST_BCY:=Y;
    LAST_BCZ:=Z;
    IF Y >= NEW_TEST_Y THEN CLIP_TOP(OP,X,Y,Z);
END;
PROCEDURE CLIP_TOP(OP,X,Y,Z);INTEGER OP;REAL X,Y,Z;
BEGIN
    REAL NEW_TEST_Y,OLD_TEST_Y,X_CLIP,Y_CLIP,Z_CLIP;
    NEW_TEST_Y:=WYH+YHM*Z;
    OLD_TEST_Y:=WYH+YHM*LAST_TCZ;
    IF (Y <= NEW_TEST_Y AND LAST_TCY > OLD_TEST_Y) OR
    (Y >= NEW_TEST_Y AND LAST_TCY < OLD_TEST_Y) THEN
    BEGIN
        Z_CLIP:=((LAST_TCZ)*(Y-LAST_TCY)+(Z-LAST_TCZ)*(WYH-LAST_TCY
        ))
        /((Y-LAST_TCY)-(Z-LAST_TCZ)*(YHM));
        Y_CLIP:=WYH+YHM*Z_CLIP;
        X_CLIP:=GET_X_CLIP(LAST_TCX,LAST_TCY,LAST_TCZ,X,Y,Z,Z_CLIP,
        Y_CLIP);
        IF LAST_TCY > OLD_TEST_Y THEN
        CLIP_REAR(1,X_CLIP,Y_CLIP,Z_CLIP) ELSE
        CLIP_REAR(OP,X_CLIP,Y_CLIP,Z_CLIP);
    END;
    LAST_TCX:=X;
    LAST_TCY:=Y;
    LAST_TCZ:=Z;
    IF Y <= NEW_TEST_Y THEN CLIP_REAR(OP,X,Y,Z);
END;
REAL PROCEDURE GET_X_CLIP(X1,Y1,Z1,X2,Y2,Z2,Z_CLIP,Y_CLIP);
REAL X1,Y1,Z1,X2,Y2,Z2,Z_CLIP,Y_CLIP;
BEGIN
    IF ABS(Z2-Z1) > EPSILON THEN
    GET_X_CLIP:=X1+((X2-X1)*(Z_CLIP-Z1)/(Z2-Z1))
    ELSE IF ABS(Y2-Y1) > EPSILON THEN
    GET_X_CLIP:=X1+((X2-X1)*(Y_CLIP-Y1)/(Y2-Y1))
    ELSE GET_X_CLIP:=X1;
END;
PROCEDURE CLIP_REAR(OP,X,Y,Z);INTEGER OP;REAL X,Y,Z;
BEGIN
```

```
     REAL X_CLIP,Y_CLIP,Z_CLIP,DEL_Z;
     IF REAR_FLAG THEN
     BEGIN
          IF (Z >= RZ_CLIP AND LAST_RRCZ < RZ_CLIP) OR
          (Z <= RZ_CLIP AND LAST_RRCZ > RZ_CLIP) THEN
          BEGIN
               DEL_Z:=(RZ_CLIP-LAST_RRCZ)/(Z-LAST_RRCZ);
               X_CLIP:=LAST_RRCX+DEL_Z*(X-LAST_RRCX);
               Y_CLIP:=LAST_RRCY+DEL_Z*(Y-LAST_RRCY);
               IF LAST_RRCZ < RZ_CLIP THEN
               CLIP_FRONT(OP,X_CLIP,Y_CLIP,RZ_CLIP) ELSE
               CLIP_FRONT(OP,X_CLIP,Y_CLIP,RZ_CLIP);
          END;
          LAST_RRCX:=X;
          LAST_RRCY:=Y;
          LAST_RRCZ:=Z;
          IF Z >= RZ_CLIP THEN CLIP_FRONT(OP,X,Y,Z);
     END
     ELSE
     CLIP_FRONT(OP,X,Y,Z);
END;
PROCEDURE CLIP_FRONT(OP,X,Y,Z);INTEGER OP;REAL X,Y,Z;
BEGIN
     REAL DEL_Z,X_CLIP,Y_CLIP,Z_CLIP;
     IF FRONT_FLAG THEN
     BEGIN
          IF (Z <= FZ_CLIP AND LAST_FCZ > FZ_CLIP) OR
          (Z >= FZ_CLIP AND LAST_FCZ < FZ_CLIP) THEN
          BEGIN
               DEL_Z:=(FZ_CLIP-LAST_FCZ)/(Z-LAST_FCZ);
               X_CLIP:=LAST_FCX+DEL_Z*(X-LAST_FCX);
               Y_CLIP:=LAST_FCY+DEL_Z*(Y-LAST_FCY);
               IF LAST_FCZ > FZ_CLIP THEN
               STORE_INSTRUCTION(1,X_CLIP,Y_CLIP,FZ_CLIP) ELSE
               STORE_INSTRUCTION(OP,X_CLIP,Y_CLIP,FZ_CLIP);
          END;
          LAST_FCX:=X;
          LAST_FCY:=Y;
          LAST_FCZ:=Z;
          IF Z <= FZ_CLIP THEN STORE_INSTRUCTION(OP,X,Y,Z);
     END
     ELSE
     STORE_INSTRUCTION(OP,X,Y,Z);
END;
PROCEDURE STORE_INSTRUCTION(OP,X,Y,Z);
INTEGER OP;
REAL X,Y,Z;
BEGIN
     IF PFLAG THEN
     BEGIN
          COUNT_OUT:=COUNT_OUT+1;
          IF COUNT_OUT < 33 THEN
          BEGIN
               OPB[COUNT_OUT]:=OP;
               XB[COUNT_OUT]:=X;
               YB[COUNT_OUT]:=Y;
               ZB[COUNT_OUT]:=Z;
          END;
     END
     ELSE
     BEGIN
          DO_PROJECTION(OP,X,Y,Z);
     END;
END;
PROCEDURE DO_PROJECTION(OP,X,Y,Z);INTEGER  OP;
REAL X,Y,Z;
BEGIN
     REAL  PX,PY;
     IF PERSPECTIVE_FLAG THEN
     BEGIN
          PX:=(X*SVZ-Z*SVX)/(SVZ-Z);
          PY:=(Y*SVZ-Z*SVY)/(SVZ-Z);
     END
     ELSE
     BEGIN
          PX:=(X-Z*SVX/SVZ);
          PY:=(Y-Z*SVY/SVZ);
     END;
     DO_VIEWING_TRANSFORMATION(OP,PX,PY);
END;
PROCEDURE NEW_VIEWING_CONSTANTS;
BEGIN
     WVSX:=(VXH-VXL)/(WXH-WXL);
     WVSY:=(VYH-VYL)/(WYH-WYL);
     WVTX:=VXL-WXL*WVSX;
     WVTY:=VYL-WYL*WVSY;
END;
PROCEDURE DO_VIEWING_TRANSFORMATION(OP,X,Y);INTEGER OP;
REAL X,Y;
```

```
BEGIN
     REAL VTX,VTY;
     VTX:=WVSX*X+WVTX;
     VTY:=WVSY*Y+WVTY;
     SIZE[CURRENTLY_OPEN]:=SIZE[CURRENTLY_OPEN]+1;
     PUT_INSTRUCTION(OF_FREE,OP,VTX,VTY);
     OF_FREE:=OF_FREE+1;
END;
PROCEDURE SHOW_LINE(X1,Y1,X2,Y2);REAL X1,Y1,X2,Y2;
BEGIN
     INTEGER XI,YI;
     OUTCHAR(CHAR(29));
     XI:=X1+0.5;YI:=Y1+0.5;
     TRANSMIT_COORDINATES(XI,YI);
     XI:=X2+0.5;YI:=Y2+0.5;
     TRANSMIT_COORDINATES(XI,YI);
     OUTCHAR(CHAR(31));
     BREAKOUTIMAGE;
END;
PROCEDURE TRANSMIT_COORDINATES(X,Y);INTEGER X,Y;
BEGIN
     OUTCHAR(CHAR(Y//32+32));
     OUTCHAR(CHAR(MOD(Y,32)+96));
     OUTCHAR(CHAR(X//32+32));
     OUTCHAR(CHAR(MOD(X,32)+64));
END;
BOOLEAN PROCEDURE IS_BACK_FACE;
BEGIN
     INTEGER LM,I,V1,V2;
     REAL SDX,SDY,SDZ,
     XV1,YV1,ZV1,
     XV2,YV2,ZV2,
     NX,NY,NZ,
     DIR;
     IS_BACK_FACE:=TRUE;
     LM:=1;
     FOR I:=2 STEP 1 UNTIL COUNT_OUT DO
     BEGIN
          IF XB[I] < XB[LM] THEN LM:=I;
     END;
     IF PERSPECTIVE_FLAG THEN
     BEGIN
          SDX:=SVX-XB[LM];
          SDY:=SVY-YB[LM];
          SDZ:=SVZ-ZB[LM];
     END
     ELSE
     BEGIN
          SDX:=SVX;
          SDY:=SVY;
          SDZ:=SVZ;
     END;
     XV1:=0;
     YV1:=0;
     ZV1:=0;
     V1:=LM;
     WHILE ABS(XV1)+ABS(YV1)+ABS(ZV1) < EPSILON DO
     BEGIN
          IF V1=COUNT_OUT THEN V1:=1 ELSE V1:=V1+1;
          IF V1=LM THEN GO TO LAST;
          XV1:=XB[V1]-XB[LM];
          YV1:=YB[V1]-YB[LM];
          ZV1:=ZB[V1]-ZB[LM];
     END;
     V2:=LM;
     DIR:=0;
     WHILE ABS(DIR) < EPSILON DO
     BEGIN
          IF V2= 1 THEN V2:=COUNT_OUT ELSE V2:=V2-1;
          IF V2=V1 THEN GO TO LAST;
          XV2:=XB[V2]-XB[LM];
          YV2:=YB[V2]-YB[LM];
          ZV2:=ZB[V2]-ZB[LM];
          NX:=(YV1*ZV2)-(YV2*ZV1);
          NY:=(ZV1*XV2)-(ZV2*XV1);
          NZ:=(XV1*YV2)-(XV2*YV1);
          DIR:=(SDX*NX)+(SDY*NY)+(SDZ*NZ);
     END;
     IF DIR > 0 THEN IS_BACK_FACE:=FALSE;
LAST: END;
PROCEDURE TRANSFER_TO_C(OP,X,Y,Z);
INTEGER OP;
REAL X,Y,Z;
BEGIN
     REAL PX,PY;
     IF PERSPECTIVE_FLAG THEN
     BEGIN
          PX:=(X*SVZ-Z*SVX)/(SVZ-Z);
          PY:=(Y*SVZ-Z*SVY)/(SVZ-Z);
```

```
     END
     ELSE
     BEGIN
          PX:=(X-Z*SVX/SVZ);
          PY:=(Y-Z*SVY/SVZ);
     END;
     PUT_IN_C(OP,PX,PY,Z);
END;
PROCEDURE PUT_IN_C(OP,X,Y,Z);
INTEGER OP;
REAL X,Y,Z;
BEGIN
     OPC[CFREE]:=OP;
     XC[CFREE]:=X;
     YC[CFREE]:=Y;
     ZC[CFREE]:=Z;
     CFREE:=CFREE+1;
END;
PROCEDURE ELIMINATE_HIDING;
BEGIN
     INTEGER NUM;
     IF CFREE > 1 THEN
     BEGIN
          SPLIT_INTO_TRIANGLES(NUM);
          COMPARE_ALL_TRIANGLES(NUM);
          SORT$SAVE#TRIANGLES(NUM);
     END;
END;
PROCEDURE SPLIT_INTO_TRIANGLES(NUM);NAME NUM;INTEGER NUM;
!-------------------------------------------------------------------
the "c-buffer" constitutes arrays "opc,xc,yc,zc"
this procedure extracts polygons one by one from
"c-buffer",places polygon temporarily in "b-buffer"
,splits the polygon in the "b-buffer" into triangles
and places these triangles into the "d-buffer".

global variables: opc,xc,yc,zc,cfree
------------------------------------------------------------------;
BEGIN
     INTEGER I,J,NO_SIDES;
     I:=1;
     DFREE:=1;
     WHILE I LE CFREE DO
     BEGIN
          NO_SIDES:=OPC[I];
          BFREE:=1;
          FOR J:=1 STEP 1 UNTIL NO_SIDES DO
          BEGIN
               I:=I+1;
               PUT_IN_B(OPC[I],XC[I],YC[I],ZC[I]);
          END;
          SPLIT_POLYGON(NO_SIDES);
          I:=I+1;
     END;
     NUM:=(DFREE-1)//3;
END;
PROCEDURE PUT_IN_B(OP,X,Y,Z);INTEGER OP;REAL X,Y,Z;
!---------------------------------------------------------------------
this procedure loads the instruction sent through its
parameters into the free location of "b-buffer".
the "b-buffer" constitutes arrays "opb,xb,yb,zb".

global variables: opb,xb,yb,zb,bfree
--------------------------------------------------------------------;
BEGIN
     OPB[BFREE]:=OP;
     XB[BFREE]:=X;
     YB[BFREE]:=Y;
     ZB[BFREE]:=Z;
     BFREE:=BFREE+1;
END;
PROCEDURE SPLIT_POLYGON(NO_SIDES);INTEGER NO_SIDES;
!------------------------------------------------------------------
this procedure splits the polygon stored in the
"b-buffer" into triangles and these triangles
are stored into the "d-buffer".
-----------------------------------------------------------------;
BEGIN
     INTEGER I,OPSAVE;
     OPSAVE:=OPB[1];
     OPB[1]:=1;
     FOR I:=3 STEP 1 UNTIL NO_SIDES DO
     BEGIN
          IF I=NO_SIDES THEN OPB[I-2]:=OPSAVE;
          IF NOT IN_LINE(I) THEN PUT_IN_D(I-2,I);
          SHIFT_INSTRUCTION(I-2,I-1);
          OPB[I]:=1;
     END;
END;
```

```
BOOLEAN PROCEDURE IN_LINE(I3);INTEGER I3;
BEGIN
     INTEGER I2,I1;
     I2:=I3-1;
     I1:=I3-2;
     IN_LINE:=ABS(((XB[I2]-XB[I1])*(YB[I2]-YB[I3]))-((XB[I2]
     -XB[
     I3])*(YB[I2]-YB[I1
     ]))) LE EPSILON;
END;

PROCEDURE PUT_IN_D(M,N);INTEGER M,N;
BEGIN
     INTEGER I;
     FOR I:=M STEP 1 UNTIL N DO
     BEGIN
          OPD[DFREE]:=OPB[I];
          XD[DFREE]:=XB[I];
          YD[DFREE]:=YB[I];
          ZD[DFREE]:=ZB[I];
          DFREE:=DFREE+1;
     END;
END;
PROCEDURE SHIFT_INSTRUCTION(N1,N2);INTEGER N1,N2;
BEGIN
     OPB[N2]:=OPB[N1];
     XB[N2]:=XB[N1];
     YB[N2]:=YB[N1];
     ZB[N2]:=ZB[N1];
END;
PROCEDURE COMPARE_ALL_TRIANGLES(NUM);
INTEGER NUM;
BEGIN
     INTEGER I,J,K,NUM_1,COMPARE;
     FREE:=1;
     FOR I:=1 STEP 1 UNTIL NUM DO
     BEGIN
          BACK_TOTAL[I]:=0;
          START_INDEX[I]:=0;
     END;
     NUM_1:=NUM-1;
     FOR I:=1 STEP 1 UNTIL (NUM_1) DO
     FOR J:=(I+1) STEP 1 UNTIL NUM DO
     BEGIN
          COMPARE:=COMPARE_TRIANGLES(3*I,3*J);
          IF COMPARE < 0 THEN ADD_FIRST_TO_FRONT_LIST_OF_SECOND(I,J);
          IF COMPARE > 0 THEN ADD_FIRST_TO_FRONT_LIST_OF_SECOND(J,I);
     END;
END;
INTEGER PROCEDURE COMPARE_TRIANGLES(LVA,LVB);
INTEGER LVA,LVB;
BEGIN
     REAL ARRAY TA[1:5,1:3],TB[1:5,1:3];
     INTEGER COMPARE,I1,I2;
     REAL XMINA,YMINA,XMAXA,YMAXA,XMINB,YMINB,XMAXB,YMAXB;
     FILARRAY(TA,LVA);
     FILARRAY(TB,LVB);
     XMINA:=TA[4,1];
     YMINA:=TA[4,2];
     XMAXA:=TA[5,1];
     YMAXA:=TA[5,2];
     XMINB:=TB[4,1];
     YMINB:=TB[4,2];
     XMAXB:=TB[5,1];
     YMAXB:=TB[5,2];
     COMPARE:=0;
     IF (XMAXA-EPSILON) < XMINB OR
     (YMAXA-EPSILON) < YMINB OR
     (XMINA+EPSILON) > XMAXB OR
     (YMINA+EPSILON) > YMAXB THEN
     GO TO LAST;
     I1:=1;I2:=2;
     WHILE (COMPARE=0 AND I1<=3) DO BEGIN
          COMPARE:=COMPARE_SIDE(I1,I2,TA,TB);
          I1:=I1+1;
          IF I1=3 THEN I2:=1 ELSE I2:=I2+1;
     END;
     IF COMPARE EQ 0 THEN COMPARE:=COMPARE_IF_CONTAINED(TA,TB);
     LAST:COMPARE_TRIANGLES:=COMPARE;
END;
PROCEDURE FILARRAY(ARY,IDX);REAL ARRAY ARY;INTEGER IDX;
BEGIN
     INTEGER I1,I2;
     REAL XMIN,XMAX,YMIN,YMAX,ZMIN,ZMAX,TEMP;
     FOR I1:=1 STEP 1 UNTIL 3 DO
     BEGIN
          I2:=IDX-3+I1;
          ARY[I1,1]:=XD[I2];
          ARY[I1,2]:=YD[I2];
```

```
            ARY[J1,3]:=ZD[I2];
        END;
        XMIN:=XMAX:=XD[IDX];
        YMIN:=YMAX:=YD[IDX];
        ZMIN:=ZMAX:=ZD[IDX];
        FOR I1:=1 STEP 1 UNTIL 2 DO
        BEGIN
            I2:=IDX-3+I1;
            TEMP:=XD[I2];
            IF TEMP < XMIN THEN XMIN:=TEMP ELSE IF TEMP > XMAX THEN
            XMAX:=TEMP;
            TEMP:=YD[I2];
            IF TEMP < YMIN THEN YMIN :=TEMP ELSE IF TEMP > YMAX THEN
            YMAX:=TEMP;
            TEMP:=ZD[I2];
            IF TEMP < ZMIN THEN ZMIN:=TEMP ELSE IF TEMP > ZMAX THEN
            ZMAX:=TEMP;
        END;
        ARY[4,1]:=XMIN;
        ARY[4,2]:=YMIN;
        ARY[4,3]:=ZMIN;
        ARY[5,1]:=XMAX;
        ARY[5,2]:=YMAX;
        ARY[5,3]:=ZMAX;
    END;
    INTEGER PROCEDURE COMPARE_SIDE(I1,I2,TA,TB);
    INTEGER I1,I2;REAL ARRAY TA,TB;
    BEGIN
        REAL XMINS,YMINS,XMAXS,YMAXS,
        XMINJS,XMAXJS,YMINJS,YMAXJS,
        X1A,Y1A,Z1A,X2A,Y2A,Z2A,
        X1B,Y1B,Z1B,X2B,Y2B,Z2B,
        C1,C2,D,X,Y,ZS,ZJS;
        INTEGER J1,J2,COMPARE,REL_DEP;
        XMINS:=X1A:=TA[I1,1];
        XMAXS:=X2A:=TA[I2,1];
        IF XMINS > XMAXS THEN
        BEGIN
            XMAXS:=X1A;
            XMINS:=X2A;
        END;
        YMINS:=Y1A:=TA[I1,2];
        YMAXS:=Y2A:=TA[I2,2];
        IF YMINS > YMAXS THEN
        BEGIN
            YMAXS:=Y1A;
            YMINS:=Y2A;
        END;
        COMPARE:=0;
        J1:=1;J2:=2;
        WHILE COMPARE = 0 AND J1 <= 3 DO
        BEGIN
            XMINJS:=X1B:=TB[J1,1];
            XMAXJS:=X2B:=TB[J2,1];
            IF XMINJS > XMAXJS THEN
            BEGIN
                XMAXJS:=X1B;
                XMINJS:=X2B;
            END;
            YMINJS:=Y1B:=TB[J1,2];
            YMAXJS:=Y2B:=TB[J2,2];
            IF YMINJS > YMAXJS THEN
            BEGIN
                YMAXJS:=Y1B;
                YMINJS:=Y2B;
            END;
            IF (XMAXS-EPSILON) < XMINJS OR
            (YMAXS-EPSILON) < YMINJS OR
            (XMINS+EPSILON) > XMAXJS OR
            (YMINS+EPSILON) > YMAXJS  THEN
            GO TO NEXT_COMPARISION;
            D:=((X1A-X2A)*(Y1B-Y2B))-((X1B-X2B)*(Y1A-Y2A));
            IF ABS(D) > EPSILON THEN
            BEGIN
                C1:=(X1A*Y2A)-(X2A*Y1A);
                C2:=(X1B*Y2B)-(X2B*Y1B);
                X:=(((X2A-X1A)*C2)-((X2B-X1B)*C1))/D;
                Y:=(((Y2A-Y1A)*C2)-((Y2B-Y1B)*C1))/D;
                IF (X > ( XMINS+EPSILON) AND
                X < ( XMAXS-EPSILON)) OR
                (           X > (XMINJS+EPSILON) AND
                X < (XMAXJS-EPSILON)) OR
                (           Y > ( YMINS+EPSILON) AND
                Y < ( YMAXS-EPSILON)) OR
                (           Y > (YMINJS+EPSILON) AND
                Y < (YMAXJS-EPSILON)) THEN
                BEGIN
                    REL_DEP:=DEPTH_TEST(TA,TB);
                    IF REL_DEP \= 0 THEN
```

```
                BEGIN
                    COMPARE:=REL_DEP;
                END
                ELSE
                BEGIN
                    Z1A:=TA[I1,3];
                    Z2A:=TA[I2,3];
                    IF (XMAXS-XMINS) < EPSILON
                    THEN
                    ZS:=Z1A+(Z2A-Z1A)*(Y-Y1A)/(Y2A-Y1A)
                    ELSE
                    ZS:=Z1A+((Z2A-Z1A)*(X-X1A)/(X2A-X1A));
                    Z1B:=TB[J1,3];
                    Z2B:=TB[J2,3];
                    IF (XMAXJS-XMINJS) < EPSILON
                    THEN
                    ZJS:=Z1B+(Z2B-Z1B)*(Y-Y1B)/(Y2B-Y1B)
                    ELSE
                    ZJS:=Z1B+((Z2B-Z1B)*(X-X1B)/(X2B-X1B));
                    IF ABS(ZS-ZJS) > EPSILON THEN
                    BEGIN
                        IF ZJS > ZS
                        THEN COMPARE:=1
                        ELSE COMPARE:=-1;
                    END
                    ELSE
                    BEGIN
                        COMPARE:=COMPARE_AT_ENDS(I1,I2,TA,TB);
                        IF COMPARE = 0 THEN
                        COMPARE:=-COMPARE_AT_ENDS(J1,J2,TB,TA);
                    END;
                END;
            END;
        END;
        NEXT_COMPARISION: J1:=J1+1;
        IF J1=3 THEN J2:=1 ELSE J2:=J2+1;
    END;
    COMPARE_SIDE:=COMPARE;
END;
INTEGER PROCEDURE DEPTH_TEST(TA,TB);
REAL ARRAY TA,TB;
BEGIN
    REAL ZMINA,ZMAXA,ZMINB,ZMAXB;
    ZMINA:=TA[4,3];

    ZMAXA:=TA[5,3];
    ZMINB:=TB[4,3];
    ZMAXB:=TB[5,3];


    IF (ZMINB -ZMAXA)    > EPSILON THEN DEPTH_TEST:=1
    ELSE IF (ZMINA-ZMAXB) > EPSILON THEN DEPTH_TEST:=-1
    ELSE DEPTH_TEST:=0;
END;
INTEGER PROCEDURE COMPARE_AT_ENDS(I1,I2,TA,TB);
INTEGER I1,I2;
REAL ARRAY TA,TB;
BEGIN
    REAL X1,Y1,Z1,X2,Y2,Z2;
    X1:=TA[I1,1];Y1:=TA[I1,2];Z1:=TA[I1,3];
    X2:=TA[I2,1];Y2:=TA[I2,2];Z2:=TA[I2,3];
    IF INSIDE(X1,Y1,TB)  THEN
    COMPARE_AT_ENDS:=COMPARE_POINT(X1,Y1,Z1,TB)
    ELSE IF INSIDE(X2,Y2,TB) THEN
    COMPARE_AT_ENDS:=COMPARE_POINT(X2,Y2,Z2,TB)
    ELSE COMPARE_AT_ENDS:=0;
END;
BOOLEAN  PROCEDURE INSIDE(X,Y,T);REAL X,Y;REAL ARRAY T;
BEGIN
    REAL X1,Y1,X2,Y2,X3,Y3;
    INSIDE:=FALSE;
    IF (X-T[5,1]) > EPSILON OR
    T[4,1]-X > EPSILON OR
    Y-T[5,2] > EPSILON OR
    T[4,2]-Y  > EPSILON  THEN GO TO LAST;
    X1:=T[1,1];Y1:=T[1,2];
    X2:=T[2,1];Y2:=T[2,2];
    X3:=T[3,1];Y3:=T[3,2];
    IF SIGN_TEST(X,Y,X1,Y1,X2,Y2) \=
    SIGN_TEST(X3,Y3,X1,Y1,X2,Y2) THEN GO TO LAST;
    IF SIGN_TEST(X,Y,X2,Y2,X3,Y3) \=
    SIGN_TEST(X1,Y1,X2,Y2,X3,Y3) THEN GO TO LAST;
    INSIDE:=SIGN_TEST(X,Y,X3,Y3,X1,Y1)=SIGN_TEST(X2,Y2,X3,Y3,
    X1,
    Y1);
    LAST:
END;
INTEGER PROCEDURE SIGN_TEST(X,Y,X1,Y1,X2,Y2);
REAL X1,X2,Y1,Y2,X,Y;
```

```
BEGIN
    REAL A,B;
    A:=(X-X1)*(Y2-Y1);
    B:=(Y-Y1)*(X2-X1);
    IF (A - B) > EPSILON THEN SIGN_TEST:=1 ELSE
    IF (A-B) < EPSILON  THEN SIGN_TEST:=-1 ELSE
    SIGN_TEST:=0;
END;
INTEGER PROCEDURE COMPARE_POINT(X,Y,Z,T2);
REAL X,Y,Z;REAL ARRAY T2;
BEGIN
    REAL X1,Y1,Z1,
    X2,Y2,Z2,
    X3,Y3,Z3,
    A,B,C,ZT2;
    COMPARE_POINT:=0;
    X1:=T2[1,1];
    Y1:=T2[1,2];
    Z1:=T2[1,3];
    X2:=T2[2,1];
    Y2:=T2[2,2];
    Z2:=T2[2,3];
    X3:=T2[3,1];
    Y3:=T2[3,2];
    Z3:=T2[3,3];
    A:=((Y1-Y2)*(Z3-Z2))-((Y3-Y2)*(Z1-Z2));
    B:=((Z1-Z2)*(X3-X2))-((Z3-Z2)*(X1-X2));
    C:=((X1-X2)*(Y3-Y2))-((X3-X2)*(Y1-Y2));
    ZT2:=Z3-(((A*(X-X3))+(B*(Y-Y3)))/C);
    IF (ZT2-Z) > EPSILON THEN COMPARE_POINT:=1;
    IF (Z-ZT2) > EPSILON THEN COMPARE_POINT:=-1;
END;
INTEGER PROCEDURE COMPARE_IF_CONTAINED(T1,T2);
REAL ARRAY T1,T2;
BEGIN
    REAL XMIN1,XMAX1,YMIN1,YMAX1,
    XMIN2,XMAX2,YMIN2,YMAX2,
    XM,YM,ZM,EPSILON;
    INTEGER REL_DEP;
    COMPARE_IF_CONTAINED:=0;
    EPSILON:=0.0001;
    XMIN1:=T1[4,1];
    XMAX1:=T1[5,1];
    YMIN1:=T1[4,2];
    YMAX1:=T1[5,2];
    XMIN2:=T2[4,1];
    XMAX2:=T2[5,1];
    YMIN2:=T2[4,2];
    YMAX2:=T2[5,2];
    IF XMAX1 < (XMAX2+EPSILON) AND
    XMIN1 > (XMIN2-EPSILON) AND
    YMAX1 < (YMAX2+EPSILON) AND
    YMIN1 > (YMIN2-EPSILON) THEN
    BEGIN
        XM:=(T1[1,1]+T1[2,1]+T1[3,1])/3;
        YM:=(T1[1,2]+T1[2,2]+T1[3,2])/3;
        ZM:=(T1[1,3]+T1[2,3]+T1[3,3])/3;
        IF INSIDE(XM,YM,T2) THEN
        BEGIN
            REL_DEP:=DEPTH_TEST(T1,T2);
            IF REL_DEP \= 0 THEN COMPARE_IF_CONTAINED:=REL_DEP ELSE
            COMPARE_IF_CONTAINED:=COMPARE_POINT(XM,YM,ZM,T2);
        END;
    END
    ELSE
    IF XMAX2 < (XMAX1+EPSILON) AND
    XMIN2 > (XMIN1-EPSILON) AND
    YMAX2 < (YMAX1+EPSILON) AND
    YMIN2 > (YMIN1-EPSILON) THEN
    BEGIN
        XM:=(T2[1,1]+T2[2,1]+T2[3,1])/3;
        YM:=(T2[1,2]+T2[2,2]+T2[3,2])/3;
        ZM:=(T2[1,3]+T2[2,3]+T2[3,3])/3;
        IF INSIDE(XM,YM,T1) THEN
        BEGIN
            REL_DEP:=DEPTH_TEST(T1,T2);
            IF REL_DEP \= 0 THEN COMPARE_IF_CONTAINED:=REL_DEP ELSE
            COMPARE_IF_CONTAINED:=-COMPARE_POINT(XM,YM,ZM,T1);
        END;
    END;
END;
PROCEDURE ADD_FIRST_TO_FRONT_LIST_OF_SECOND(T1,T2);
INTEGER T1,T2;
BEGIN
    INFO[FREE]:=T1;
    LINK[FREE]:=START_INDEX[T2];
    START_INDEX[T2]:=FREE;
    FREE:=FREE+1;
    BACK_TOTAL[T1]:=BACK_TOTAL[T1]+1;
```

```
END;
PROCEDURE SORT#SAVE#TRIANGLES(NUMBER_OF_TRIANGLES);
INTEGER NUMBER_OF_TRIANGLES;
BEGIN
     INTEGER I,TEST_TRIANGLE,FRONT_TRIANGLE,INDEX;
     TEST_TOTAL:=0;
     FOR I:=1 STEP 1 UNTIL NUMBER_OF_TRIANGLES DO
     BEGIN
          IF BACK_TOTAL[I]=0 THEN
          BEGIN
               TEST_TOTAL:=TEST_TOTAL+1;
               TO_BE_TESTED[TEST_TOTAL]:=I;
          END;
     END;
     WHILE TEST_TOTAL \= 0 DO
     BEGIN
          TEST_TRIANGLE:=TO_BE_TESTED[TEST_TOTAL];
          BACK_TOTAL[TEST_TRIANGLE]:=-1;
          TEST_TOTAL:=TEST_TOTAL-1;
          INDEX:=START_INDEX[TEST_TRIANGLE];
          WHILE INDEX \= 0 DO
          BEGIN
               FRONT_TRIANGLE:=INFO[INDEX];
               BACK_TOTAL[FRONT_TRIANGLE]:=BACK_TOTAL[FRONT_TRIANGLE]-1;
               IF BACK_TOTAL[FRONT_TRIANGLE] = 0 THEN
               BEGIN
                    TEST_TOTAL:=TEST_TOTAL+1;
                    TO_BE_TESTED[TEST_TOTAL]:=FRONT_TRIANGLE;
               END;
               INDEX:=LINK[INDEX];
          END;
          CHECK_SIDES(TEST_TRIANGLE);
     END;
END;
PROCEDURE CHECK_SIDES(TRIANGLE);INTEGER TRIANGLE;
BEGIN
     INTEGER I,SIDES,IDX,INDEX,NEXT_INDEX,OP;
     BOOLEAN INTERSECT;
     REAL X,Y;
     REAL ARRAY T1[1:5,1:3];
     PROCEDURE PUT_TRIANGLE_IN_C;
     BEGIN
          I:=3*TRIANGLE;
          PUT_IN_C(OPD[I],XD[I],YD[I],START_INDEX[TRIANGLE]);
          PUT_IN_C(OPD[I-1],XD[I-1],YD[I-1],START_INDEX[TRIANGLE]);
          PUT_IN_C(OPD[I-2],XD[I-2],YD[I-2],START_INDEX[TRIANGLE]);
          XS:=XD[I];
          YS:=YD[I];
     END;
     PROCEDURE POP_C;
     BEGIN
          I:=CFREE-1;
          OP:=OPC[I];
          XF:=XC[I];
          YF:=YC[I];
          INDEX:=ZC[I];
     END;
     CFREE:=BFREE:=1;
     PUT_TRIANGLE_IN_C;
     WHILE CFREE \= 1 DO
     BEGIN
          POP_C;
          WHILE INDEX \= 0 AND OP \= 1 DO
          BEGIN
               IDX:=3*(INFO[INDEX]);
               FILARRAY(T1,IDX);
               NEXT_INDEX:=LINK[INDEX];
               IF INSIDE_OF(XS,YS,T1) THEN
               BEGIN
                    IF INSIDE_OF(XF,YF,T1) THEN
                    BEGIN
                         OPC[CFREE-1]:=1;
                    END
                    ELSE
                    BEGIN
                         SINGLE_INTERSECTION(X,Y,INTERSECT,IDX);
                         IF (ABS(XF-X)+ABS(YF-Y)) < EPSILON THEN
                         BEGIN
                              OPC[CFREE-1]:=1;
                         END
                         ELSE
                         BEGIN
                              PUT_IN_B(1,X,Y,0);
                              XS:=X;
                              YS:=Y;
                              ZC[CFREE-1]:=NEXT_INDEX
                         END;
                    END;
```

```
                ELSE
                BEGIN
                     IF INSIDE_OF(XF,YF,T1) THEN
                     BEGIN
                          SINGLE_INTERSECTION(X,Y,INTERSECT.IDX);
                          OPC[CFREE-1]:=1;
                          IF (ABS(XS-X))+(ABS(YS-Y)) > EPSILON THEN
                          PUT_IN_C(2,X,Y,NEXT_INDEX);
                     END
                     ELSE
                     BEGIN
                          CHOP_MIDDLE(NEXT_INDEX,IDX);
                     END;
                END;
                POP_C;
           END;
           PUT_IN_B(OP,XF,YF,0);
           CFREE:=CFREE-1;
           XS:=XF;
           YS:=YF;
      END;
      SIDES:=BFREE-1;
      DO_VIEWING_TRANSFORMATION(SIDES,XB[SIDES],YB[SIDES]);
      FOR I:=1 STEP 1 UNTIL SIDES DO
      BEGIN
           DO_VIEWING_TRANSFORMATION(OPB[I],XB[I],YB[I]);
      END;
END;
PROCEDURE SINGLE_INTERSECTION(X,Y,INTERSECT,IDX);
NAME X,Y,INTERSECT;
REAL X,Y;BOOLEAN INTERSECT;INTEGER IDX;
BEGIN
     BOOLEAN MARGINALLY_CROSS;
     REAL X1,Y1;
     MARGINALLY_CROSS:=FALSE;
     XP:=XD[IDX-2];
     YP:=YD[IDX-2];
     XQ:=XD[IDX-1];
     YQ:=YD[IDX-1];
     CHECK_INTERSECTION(X,Y,INTERSECT);

     IF INTERSECT THEN
     BEGIN
          IF (ABS(X-XS)+ABS(Y-YS)) > EPSILON
          AND (ABS(X-XF)+ABS(Y-YF)) > EPSILON THEN
          BEGIN
               GO TO LAST;
          END
          ELSE
          BEGIN
               X1:=X;
               Y1:=Y;
               MARGINALLY_CROSS:=TRUE;
          END;
     END;
     XP:=XD[IDX];
     YP:=YD[IDX];
     CHECK_INTERSECTION(X,Y,INTERSECT);
     IF INTERSECT THEN
     BEGIN
          IF (ABS(X-XS)+ABS(Y-YS)) > EPSILON
          AND (ABS(X-XF)+ABS(Y-YF)) > EPSILON THEN
          BEGIN
               GO TO LAST;
          END
          ELSE
          BEGIN
               X1:=X;
               Y1:=Y;
               MARGINALLY_CROSS:=TRUE;
          END;
     END;
     XQ:=XD[IDX-2];
     YQ:=YD[IDX-2];
     CHECK_INTERSECTION(X,Y,INTERSECT);
     IF NOT INTERSECT AND MARGINALLY_CROSS THEN
     BEGIN
          X:=X1;
          Y:=Y1;
          INTERSECT:=TRUE;
     END;
     LAST:
END;
PROCEDURE CHECK_INTERSECTION(X,Y,INTERSECT);
NAME X,Y,INTERSECT;
REAL X,Y;
BOOLEAN INTERSECT;
!----------------------------------------------------------
this procedure finds the intersection point of two
```

```
lines. one line connects the points (xs,vs) and
(xf,yf).other line connects the points (xp,yp)
and (xq,yq).intersect is true if point of
intersection is within line segments.

global variables
xs,vs,xt,yf,xp,tp,xq,yq

-------------------------------------------------------------
BEGIN
     REAL C1,C2,D,MIN1,MIN2,MAX1,MAX2;
     INTERSECT:=TRUE;
     D:=((YQ-YP)*(XF-XS))-((XQ-XP)*(YF-YS));
     IF ABS(D) < EPSILON THEN GO TO LAST;
     C1:=(YS*XF)-(XS*YF);
     C2:=(YP*XQ)-(XP*YQ);
     X:=(((XQ-XP)*C1)-((XF-XS)*C2))/D;
     MIN1:=XS;
     MAX1:=XF;
     IF XS > XF THEN
     BEGIN
          MAX1:=XS;
          MIN1:=XF;
     END;
     MIN2:=XP;
     MAX2:=XQ;
     IF XP > XQ THEN
     BEGIN
          MAX2:=XP;
          MIN2:=XQ;
     END;
     IF X < (MIN1-EPSILON) OR
     X < (MIN2-EPSILON)  OR
     X > (MAX1+EPSILON) OR
     X > (MAX2+EPSILON) THEN
     GOTO LAST;
     Y:=(((YQ-YP)*C1)-((YF-YS)*C2))/D;
     MIN1:=YS;
     MAX1:=YF;
     IF YS > YF THEN
     BEGIN
          MAX1:=YS;
          MIN1:=YF;
     END;
     MIN2:=YP;
     MAX2:=YQ;
     IF YP > YQ THEN
     BEGIN
          MAX2:=YP;
          MIN2:=YQ;
     END;

     IF Y < (MIN1-EPSILON) OR
     Y < (MIN2-EPSILON) OR
     Y > (MAX1+EPSILON) OR

     Y >  (MAX2+EPSILON) THEN
     LAST:INTERSECT:=FALSE;
END;
PROCEDURE CHOP_MIDDLE(INDEX,IDX);INTEGER INDEX,IDX;
!-------------------------------------------------------------
with index one can get the location,in the list named
"info", where information of the triangle that will be
used for comparision after chopping the line with the
current triangle .

with idx one can get the  locations ,in the 'd' buffer
where information regarding the co_ordinates of the
current triangle  that is going to be used for chopping
the line

global variables       xs,ys,xf,yf


-------------------------------------------------------------;
BEGIN
     REAL X,Y,U,V,TEMP;
     BOOLEAN INTERSECT;
     ZC(CFREE-1):=INDEX;
     DOUBLE_INTRSECTION(X,Y,U,V,INTERSECT,IDX);
     IF INTERSECT THEN
     BEGIN
          IF (SIGN_OF(XS-XF) \= SIGN_OF(X-U))
          OR (SIGN_OF(YS-YF) \= SIGN_OF(Y-V))
          THEN
          BEGIN
               TEMP:=U;U:=X;X:=TEMP;
               TEMP:=V;V:=Y;Y:=TEMP;
          END;
```

```
        IF (ABS(X-XS)+ABS(Y-YS)) < EPSILON THEN
        BEGIN
            IF (ABS(U-XF)+ABS(V-YF)) < EPSILON THEN
            BEGIN
                OPC(CFREE-1):=1;
            END
            ELSE
            BEGIN
                PUT_IN_B(1,U,V,0);
                XS:=U;
                YS:=V;
            END;
        END
        ELSE
        BEGIN
            IF (ABS(U-XF)+ABS(V-YF)) < EPSILON THEN
            BEGIN
                OPC(CFREE-1):=1;
                PUT_IN_C(2,X,Y,INDEX);
            END
            ELSE
            BEGIN
                PUT_IN_C(1,U,V,INDEX);
                PUT_IN_C(2,X,Y,INDEX);
            END;
        END;
    END;
END;
INTEGER PROCEDURE SIGN_OF(X);REAL X;
BEGIN
    SIGN_OF:=0;
    IF X < -(EPSILON) THEN SIGN_OF:= -1 ELSE IF X > EPSILON THEN
    SIGN_OF:=1;
END;
PROCEDURE DOUBLE_INTRSECTION(X,Y,U,V,INTERSECT,IDX);
NAME X,Y,U,V,INTERSECT;
REAL X,Y,U,V;BOOLEAN INTERSECT;INTEGER IDX;
BEGIN
    XP:=XD[IDX-2];
    YP:=YD[IDX-2];
    XQ:=XD[IDX-1];
    YQ:=YD[IDX-1];
    CHECK_INTERSECTION(X,Y,INTERSECT);
    XP:=XD[IDX];
    YP:=YD[IDX];
    IF INTERSECT THEN
    BEGIN
        CHECK_INTERSECTION(U,V,INTERSECT);

        IF INTERSECT THEN
        BEGIN
            INTERSECT:=(ABS(U-X)+ABS(V-Y)) > EPSILON;
            IF INTERSECT THEN                   GO TO LAST;
        END;
    END

    ELSE
    BEGIN
        CHECK_INTERSECTION(X,Y,INTERSECT);
        IF NOT INTERSECT THEN GO TO LAST;
    END;
    XQ:=XD[IDX-2];
    YQ:=YD[IDX-2];
    CHECK_INTERSECTION(U,V,INTERSECT);
    INTERSECT:=INTERSECT AND (ABS(U-X)+ABS(V-Y)) > EPSILON;
    LAST:
END;
PROCEDURE CLEAR_SCREEN;
BEGIN
    INTEGER I,J;
    RUBOUT;
    FOR I:=1 STEP 1 UNTIL 10 DO
    FOR J:=1 STEP 1 UNTIL 60 DO
    BEGIN
        OUTCHAR(CHAR(0));
        BREAKOUTIMAGE;
    END;
END;
BOOLEAN PROCEDURE INSIDE_OF(X,Y,T);
REAL X,Y;
REAL ARRAY T;
BEGIN
    INTEGER TEMP;
    REAL X1,Y1,X2,Y2,X3,Y3,XMIN,XMAX,YMIN,YMAX;
    INSIDE_OF:=FALSE;
    XMIN:=T[4,1];
    XMAX:=T[5,1];
    YMIN:=T[4,2];
    YMAX:=T[5,2];
```

```
    IF X < (XMIN-EPSILON) OR
    Y < (YMIN-EPSILON) OR
    X > (XMAX+EPSILON) OR
    Y > (YMAX+EPSILON) THEN
    GOTO LAST;
    X1:=T[1,1]; Y1:=T[1,2];
    X2:=T[2,1]; Y2:=T[2,2];
    X3:=T[3,1]; Y3:=T[3,2];
    TEMP:=SIGN_TEST(X,Y,X1,Y1,X2,Y2);
    IF TEMP=0 THEN GOTO LAST_BUT_ONE;
    IF TEMP \= SIGN_TEST(X3,Y3,X1,Y1,X2,Y2) THEN GOTO LAST;
    TEMP:=SIGN_TEST(X,Y,X2,Y2,X3,Y3);
    IF TEMP=0 THEN GOTO LAST_BUT_ONE;
    IF TEMP \= SIGN_TEST(X1,Y1,X2,Y2,X3,Y3) THEN GOTO LAST;
    TEMP:=SIGN_TEST(X,Y,X3,Y3,X1,Y1);
    IF TEMP=0 THEN GOTO LAST_BUT_ONE;
    IF TEMP = SIGN_TEST(X2,Y2,X3,Y3,X1,Y1) THEN
    LAST_BUT_ONE:INSIDE_OF:=TRUE;
    LAST:
END;
PROCEDURE OUTMESSAGE;
BEGIN
    OUTTEXT("RESPONSE THAT YOU HAVE TYPED IN IS INAPPROPRIATE");
    OUTIMAGE;
    OUTTEXT("TYPE ONCE AGAIN 'YES' OR 'NO' IN TTY");
    OUTIMAGE;
END;
PROCEDURE INIT_SEGMENTS;
BEGIN
    DELETE_ALL_SEGMENTS;
    UPDATE_DISPLAY;
    BFREE:=CFREE:=DFREE:=1;
    SET_ITM_TO_IDENTITY;
END;
PROCEDURE MOVE_CURSOR(X,Y);
INTEGER X,Y;
BEGIN
    OUTCHAR(CHAR(29));
    TRANSMIT_COORDINATES(X,Y);
    OUTCHAR(CHAR(31));
    BREAKOUTIMAGE;
END;


MAX_LIMIT:=10;
INIT_SEGMENTS;
OUTTEXT(
"DO YOU WANT  'WIRE FRAME' TYPE DISPLAY OF THE OBJECT");
OUTIMAGE;
OUTTEXT("RESPOND BY TYPING   'YES' OR 'NO' IN TTY");
OUTIMAGE;
LOOP1:INIMAGE;
CH:=INCHAR;
IF CH='Y' THEN
BEGIN
    WIRE_FRAME:=TRUE;
    GOTO JUMP;
END
ELSE IF CH='N' THEN WIRE_FRAME:=FALSE
ELSE
BEGIN
    OUTMESSAGE;
    GOTO LOOP1;
END;
OUTTEXT("DOES THE OBJECT TO BE DISPLAYED HAS CLOSED SURFACE"
);
OUTIMAGE;
OUTTEXT("RESPOND BY TYPING  'YES' OR 'NO' IN TTY");
OUTIMAGE;
LOOP2:INIMAGE;
CH:=INCHAR;
IF CH='Y' THEN
CLOSED_SURFACE:=TRUE
ELSE
IF CH='N' THEN
BEGIN
    CLOSED_SURFACE:=FALSE;
    GOTO JUMP;
END
ELSE
BEGIN
    OUTMESSAGE;
    GOTO LOOP2;
END;
OUTTEXT(
"DOES YOUR PROGRAM IS GOING TO DISPLAY MULTIPLE OBJECTS");
OUTIMAGE;
OUTTEXT("RESPOND BY TYPING 'YES' OR 'NO' IN TTY");
```

```
OUTIMAGE;
LOOP3:INIMAGE;
CH:=INCHAR;
IF CH='Y' THEN
HIDDEN_FLAG:=TRUE
ELSE IF CH='N' THEN

BEGIN
     OUTTEXT("ARE YOU USING INSTANCE TRANSFORMATION");
     OUTTEXT("FUNCTIONS IN YOUR PROGRAM");
     OUTIMAGE;
     OUTTEXT("RESPOND BY TYPING 'YES' OR 'NO' IN TTY");
     OUTIMAGE;
     LOOP4:INIMAGE;
     CH:=INCHAR;
     IF CH='Y' THEN HIDDEN_FLAG:=TRUE
     ELSE IF CH='N' THEN HIDDEN_FLAG:=FALSE
     ELSE
     BEGIN
          OUTMESSAGE;
          GOTO LOOP4;
     END;
END
ELSE
BEGIN
     OUTMESSAGE;
     GOTO LOOP3;
END;
JUMP:CLEAR_SCREEN;
EPSILON:=.00001;
END;
```

```
!FILE NAME IS SPHERE.SIM;
BEGIN
    EXTERNAL PROCEDURE RUBOUT;
    EXTERNAL CLASS GRAS;
    GRAS BEGIN
        INTEGER NVS,NHS,R,I,J,JMINUS;
        REAL TITA,DELTITA,CT,ST,X,Y,Z;
        REAL ARRAY YIN,ZIN[0:25];
        REAL ARRAY XPRES,YPRES,ZPRES[0:25];
        REAL ARRAY XNEXT,YNEXT,ZNEXT[0:25];
        REAL ARRAY AX,AY,AZ[1:4];
        PROCEDURE SPHERE;
        BEGIN
            FOR J:=0 STEP 1 UNTIL NVS DO
            BEGIN
                DO_CURRENT_TRANSFORMATION(0,YIN[J],ZIN[J]);
                XPRES[J]:=VPTX;
                YPRES[J]:=VPTY;
                ZPRES[J]:=VPTZ;
            END;
            TITA:=0;
            FOR I:=1 STEP 1 UNTIL NHS DO
            BEGIN
                TITA:=TITA+DELTITA;
                CT:=COS(TITA);
                ST:=SIN(TITA);
                FOR J:=0 STEP 1 UNTIL NVS DO
                BEGIN
                    X:=ZIN[J]*ST;
                    Y:=YIN[J];
                    Z:=ZIN[J]*CT;
                    DO_CURRENT_TRANSFORMATION(X,Y,Z);
                    XNEXT[J]:=VPTX;
                    YNEXT[J]:=VPTY;
                    ZNEXT[J]:=VPTZ;
                END;
                FOR J:=1 STEP 1 UNTIL NVS DO
                BEGIN
                    JMINUS:=J-1;
                    AX[1]:=XPRES[JMINUS];
                    AY[1]:=YPRES[JMINUS];
                    AZ[1]:=ZPRES[JMINUS];
                    AX[2]:=XPRES[J];
                    AY[2]:=YPRES[J];
                    AZ[2]:=ZPRES[J];
                    AX[3]:=XNEXT[J];
                    AY[3]:=YNEXT[J];
                    AZ[3]:=ZNEXT[J];
                    AX[4]:=XNEXT[JMINUS];
                    AY[4]:=YNEXT[JMINUS];
                    AZ[4]:=ZNEXT[JMINUS];
                    POLYGON(AX,AY,AZ,4);
                END;
                FOR J:=0 STEP 1 UNTIL NVS DO
                BEGIN
                    XPRES[J]:=XNEXT[J];
                    YPRES[J]:=YNEXT[J];
                    ZPRES[J]:=ZNEXT[J];
                END;
            END;
        END;
        SETUP_VIEWPLANE(0,0,0,1,1,1,15,0,1,0);
        SET_PROJECTION_PARAMETERS(0,0,1,FALSE);
        SET_WINDOW(-12,12,-12,12);
        SET_VIEW_DEPTH(0,30);
        SET_VIEWPORT(0,780,0,780);
        SET_DEPTH_FLAGS(TRUE,TRUE);
        NVS:=25;
        NHS:=NVS*2;
        TITA:=0;
        DELTITA:=3.1415926535/NVS;
        R:=10;
        FOR J:=0 STEP 1 UNTIL NVS DO
        BEGIN
            YIN[J]:=R*COS(TITA);
            ZIN[J]:=R*SIN(TITA);
            TITA:=TITA+DELTITA;
        END;
        SPHERE;
        POST_SEGMENT(0);
        UPDATE_DISPLAY;
        INIMAGE;
    END;
END
```

```
!FILE NAME IS RING.SIM;
BEGIN
    EXTERNAL PROCEDURE RUBOUT;
    EXTERNAL CLASS GRAS;
    GRAS BEGIN
        INTEGER NVS,NHS,RA,RB,I,J,JPLUS;
        REAL TITA,DELTITA,CT,ST,X,Y,Z;
        REAL ARRAY YIN[1:15],ZIN[1:15],
        XPRES[1:15],YPRES[1:15],ZPRES[1:15],
        XNEXT[1:15],YNEXT[1:15],ZNEXT[1:15],
        AX[1:4],AY[1:4],AZ[1:4];
        SETUP_VIEWPLANE(0,0,0,1,1,1,25,0,1,0);
        SET_PROJECTION_PARAMETERS(0,0,1,FALSE);
        SET_DEPTH_FLAGS(TRUE,TRUE);
        SET_WINDOW(-25,25,-25,25);
        SET_VIEW_DEPTH(0,50);
        SET_VIEWPORT(0,780,0,780);
        NVS:=15;
        NHS:=2*NVS;
        RA:=15;
        RB:=5;
        TITA:=0;
        DELTITA:=6.283185307/NVS;
        FOR J:=1 STEP 1 UNTIL NVS DO
        BEGIN
            TITA:=TITA+DELTITA;
            YIN[J]:=Y:=RB*COS(TITA);
            ZIN[J]:=Z:=RA+RB*SIN(TITA);
            DO_CURRENT_TRANSFORMATION(0,Y,Z);
            XPRES[J]:=VPTX;
            YPRES[J]:=VPTY;
            ZPRES[J]:=VPTZ;
        END;
        TITA:=0;
        DELTITA:=6.283185307/NHS;
        FOR I:=1 STEP 1 UNTIL NHS DO
        BEGIN
            TITA:=TITA+DELTITA;
            CT:=COS(TITA);
            ST:=SIN(TITA);
            FOR J:=1 STEP 1 UNTIL NVS DO
            BEGIN
                X:=ZIN[J]*ST;
                Y:=YIN[J];
                Z:=ZIN[J]*CT;
                DO_CURRENT_TRANSFORMATION(X,Y,Z);
                XNEXT[J]:=VPTX;
                YNEXT[J]:=VPTY;
                ZNEXT[J]:=VPTZ;
            END;
            FOR J:=1 STEP 1 UNTIL NVS DO
            BEGIN
                IF J=NVS THEN JPLUS:=1 ELSE JPLUS:=J+1;
                AX[1]:=XPRES[J];
                AY[1]:=YPRES[J];
                AZ[1]:=ZPRES[J];
                AX[4]:=XNEXT[J];
                AY[4]:=YNEXT[J];
                AZ[4]:=ZNEXT[J];
                AX[3]:=XNEXT[JPLUS];
                AY[3]:=YNEXT[JPLUS];
                AZ[3]:=ZNEXT[JPLUS];
                AX[2]:=XPRES[JPLUS];
                AY[2]:=YPRES[JPLUS];
                AZ[2]:=ZPRES[JPLUS];
                POLYGON(AX,AY,AZ,4);
            END;
            FOR J:=1 STEP 1 UNTIL NVS DO
            BEGIN
                XPRES[J]:=XNEXT[J];
                YPRES[J]:=YNEXT[J];
                ZPRES[J]:=ZNEXT[J];
            END;
        END;
        POST_SEGMENT(0);
        UPDATE_DISPLAY;
    END;
    INIMAGE;
END
```

```
!FILE NAME IS TYRE.SIM;
BEGIN
     EXTERNAL PROCEDURE ROBOOT;
     EXTERNAL CLASS GRAS;
     GRAS BEGIN
          INTEGER RA,RB,I,J,JPLUS;
          REAL TITA,DELTITA,CT,ST,X,Y,Z;
          REAL ARRAY YIN,ZIN,
          XPRES,YPRES,ZPRES,
          XNEXT,YNEXT,ZNEXT[1:10],
          AX,AY,AZ[1:4];
          SETUP_VIEWPLANE(0,0,0,1,1,1,25,0,1,0);
          SET_PROJECTION_PARAMETERS(0,0,1,FALSE);
          SET_DEPTH_FLAGS(TRUE,TRUE);
          SET_WINDOW(-25,25,-25,25);
          SET_VIEW_DEPTH(0,50);
          SET_VIEWPORT(0,780,0,780);
          RA:=15;
          RB:=5;
          DELTITA:=6.283185307/12;
          TITA:=DELTITA;
          FOR J:=1 STEP 1 UNTIL 9 DO
          BEGIN
               TITA:=TITA+DELTITA;
               YIN[J]:=Y:=RB*SIN(TITA);
               ZIN[J]:=Z:=RA-RB*COS(TITA);
               DO_CURRENT_TRANSFORMATION(0,Y,Z);
               XPRES[J]:=VPTX;
               YPRES[J]:=VPTY;
               ZPRES[J]:=VPTZ;
          END;
          TITA:=0;
          FOR I:=1 STEP 1 UNTIL 12 DO
          BEGIN
               TITA:=TITA+DELTITA;
               CT:=COS(TITA);
               ST:=SIN(TITA);
               FOR J:=1 STEP 1 UNTIL 9 DO
               BEGIN
                    X:=ZIN[J]*ST;
                    Y:=YIN[J];
                    Z:=ZIN[J]*CT;
                    DO_CURRENT_TRANSFORMATION(X,Y,Z);
                    XNEXT[J]:=VPTX;
                    YNEXT[J]:=VPTY;
                    ZNEXT[J]:=VPTZ;
               END;
               FOR J:=1 STEP 1 UNTIL 8 DO
               BEGIN
                    JPLUS:=J+1;
                    AX[1]:=XPRES[J];
                    AY[1]:=YPRES[J];
                    AZ[1]:=ZPRES[J];
                    AX[4]:=XNEXT[J];
                    AY[4]:=YNEXT[J];
                    AZ[4]:=ZNEXT[J];
                    AX[3]:=XNEXT[JPLUS];
                    AY[3]:=YNEXT[JPLUS];
                    AZ[3]:=ZNEXT[JPLUS];
                    AX[2]:=XPRES[JPLUS];
                    AY[2]:=YPRES[JPLUS];
                    AZ[2]:=ZPRES[JPLUS];
                    POLYGON(AX,AY,AZ,4);
               END;
               FOR J:=1 STEP 1 UNTIL 9 DO
               BEGIN
                    XPRES[J]:=XNEXT[J];
                    YPRES[J]:=YNEXT[J];
                    ZPRES[J]:=ZNEXT[J];
               END;
          END;
          ELIMINATE_HIDING;
          POST_SEGMENT(0);
          UPDATE_DISPLAY;
     END;
     INIMAGE;
END
```

```
!FILE NAME:HOUSES.SIM;
!THIS PROGRAM DISPLAYS 4 HOUSES;
BEGIN
    EXTERNAL PROCEDURE RUBOUT;
    EXTERNAL CLASS GRAS;
    GRAS BEGIN
        TEXT BUF;
        INTEGER TOTAL_POINTS,TOTAL_FACES,SIDES,I,J,TEMP;
        REAL ARRAY XIN[1:59],YIN[1:59],ZIN[1:59];
        INTEGER ARRAY P[1:20,1:5];
        REAL ARRAY XA[1:59],YA[1:59],ZA[1:59];
        REAL ARRAY AX[1:5],AY[1:5],AZ[1:5];
        INTEGER ARRAY N[1:20];
        CHARACTER CH;
        PROCEDURE HOUSE;
        BEGIN
            FOR I:=1 STEP 1 UNTIL TOTAL_POINTS DO
            BEGIN
                DO_CURRENT_TRANSFORMATION(XIN[I],YIN[I],ZIN[I]);
                XA[I]:=VPTX;
                YA[I]:=VPTY;
                ZA[I]:=VPTZ;
            END;
            FOR I:=1 STEP 1 UNTIL TOTAL_FACES DO
            BEGIN
                SIDES:=N[I];
                FOR J:=1 STEP 1 UNTIL SIDES DO
                BEGIN
                    TEMP:=P[I,J];
                    AX[J]:=XA[TEMP];
                    AY[J]:=YA[TEMP];
                    AZ[J]:=ZA[TEMP];
                END;
                POLYGON(AX,AY,AZ,SIDES);
            END;
        END;
        SETUP_VIEWPLANE(0,0,11,1,1,1,32,0,1,0);
        SET_PROJECTION_PARAMETERS(0,0,1,FALSE);
        SET_DEPTH_FLAGS(TRUE,TRUE);
        SET_WINDOW(-25,25,-25,25);
        SET_VIEW_DEPTH(0,100);
        SET_VIEWPORT(0,780,0,780);
        INSPECT NEW INFILE("HOUSES.DAT") DO
        BEGIN
            BUF:-BLANKS(20);
            OPEN(BUF);
            INIMAGE;
            INIMAGE;
            TOTAL_POINTS:=ININT;
            FOR I:=1 STEP 1 UNTIL TOTAL_POINTS DO
            BEGIN
                INIMAGE;
                XIN[I]:=ININT;
                YIN[I]:=ININT;
                ZIN[I]:=ININT;
            END;
            INIMAGE;
            TOTAL_FACES:=ININT;
            FOR I:=1 STEP 1 UNTIL TOTAL_FACES DO
            BEGIN
                INIMAGE;
                SIDES:= N[I]:=ININT;
                FOR J:=1 STEP 1 UNTIL SIDES DO P[I,J]:=ININT;
            END;
            CLOSE;
        END;
        CREATE_SEGMENT(1);
        HOUSE;
        CLOSE_SEGMENT;
        CREATE_SEGMENT(2);
        ROTATE_I(2,90);
        TRANSLATE_I(-11,0,11);
        BEGIN_XFORM;
        HOUSE;
        END_XFORM;
        CLOSE_SEGMENT;
        CREATE_SEGMENT(3);
        ROTATE_I(2,180);
        TRANSLATE_I(0,0,22);
        BEGIN_XFORM;
        HOUSE;
        END_XFORM;
        CLOSE_SEGMENT;
        CREATE_SEGMENT(4);
        ROTATE_I(2,-90);
        TRANSLATE_I(11,0,11);
        BEGIN_XFORM;
        HOUSE;
        END_XFORM;
```

```
CLOSE_SEGMENT;
ELIMINATE_HIDING;
FOR I:=0 STEP 1 UNTIL 4 DO POST_SEGMENT(I);
UPDATE_DISPLAY;
LOOP1:MOVE_CURSOR(800,390);
OUTTEXT("PRESS <CR> TO");
OUTIMAGE;
MOVE_CURSOR(800,370);
OUTTEXT("CONTINUE");
OUTIMAGE;
INIMAGE;
CLEAR_SCREEN;
FOR I:=0 STEP 1 UNTIL 4 DO UNPOST_SEGMENT(I);
OUTTEXT("DO YOU WANT MODIFICATIONS IN THE DISPLAY");
OUTIMAGE;
OUTTEXT("TYPE 'YES' OR 'NO' IN TTY");
OUTIMAGE;
LOOP2:INIMAGE;
CH:=INCHAR;
IF CH = 'Y' THEN
BEGIN
     OUTTEXT("HOW MANY SEGMENTS DO YOU WANT TO ");
     OUTTEXT("BE DISPLAYED");
     OUTIMAGE;
     INIMAGE;
     TEMP:=ININT;

     OUTTEXT("TYPE THE NAMES OF THE SEGMENTS THAT ");
     OUTTEXT("ARE TO BE DISPLAYED");
     OUTIMAGE;
     INIMAGE;
     FOR I:=1 STEP 1 UNTIL TEMP DO POST_SEGMENT(ININT);
     UPDATE_DISPLAY;
     DRAW_VIEWPORT;
     GOTO LOOP1;
END
ELSE
BEGIN
     IF CH='N' THEN GOTO LAST
     ELSE
     BEGIN
          OUTMESSAGE;
          GOTO LOOP2;

     END;
END;
LAST:
```

FIGE:HOUSES.DAT

59

-6 0 -4

-6 0 4

6 0 4

6 0 -4

-6 8 -4

-6 8 4

6 8 4

6 8 -4

-6 11 0

6 11 0

-4 7 4

-4 5 4

-1 5 4

-1 7 4

-4 3 4

-4 1 4

-1 1 4

-1 3 4

0 3 4

0 1 4

5 1 4

5 3 4

1 7 4

1 5 4

4 5 4

4 7 4

5 7 -4

5 5 -4

2 5 -4

2 7 -4

5 3 -4

5 1 -4

2 1 -4

2 3 -4

-2 3 -4

-2 1 -4

-5 1 -4

-5 3 -4

-2 7 -4

-2 5 -4

-5 5 -4

-5 7 -4

0 1 -4

1 3 -4

1 0 -4

-1 0 -4

-1 3 -4

6 7 -1

6 5 -1

6 5 -3

6 7 -3

-6 7 1

-6 5 1

-6 5 3

-6 7 3

-6 3 1

-6 0 1

-6 0 3

-6 3 3

20

4 4 3 2 1

4 6 2 3 7

4 9 6 7 10

4 5 9 10 8

4 1 5 8 4

4 2 1 4 3

5 10 7 3 4 8

5 2 6 9 5 1

4 11 12 13 14

4 15 16 17 18

4 19 20 21 22

4 23 24 25 26

4 27 28 29 30

4 31 32 33 34

4 35 36 37 38

4 39 40 41 42

5 43 44 45 46 47

4 48 49 50 51

4 52 53 54 55

4 56 57 58 59

```
!FILE NAME:HOLLOW.SIM;
!THIS PROGRAM DISPLAYS HOLLOW CUBE MADE BY INSTANT
TRANSFORMATIONS OF A SOLID CUBE;
BEGIN
    EXTERNAL PROCEDURE RUBOUT;
    EXTERNAL CLASS GRAS;
    GRAS BEGIN
        TEXT BUF;
        INTEGER TOTAL_POINTS,TOTAL_FACES,SIDES,I,J,TEMP;
        REAL ARRAY XIN,YIN,ZIN[1:8];
        INTEGER ARRAY P[1:6,1:4];
        REAL X,Y,Z,NX,NY,NZ;
        REAL ARRAY AX,AY,AZ[1:4];
        PROCEDURE BLOCK;
        BEGIN
            FOR I:=1 STEP 1 UNTIL TOTAL_FACES DO
            BEGIN
                FOR J:=1 STEP 1 UNTIL SIDES DO
                BEGIN
                    TEMP:=P[I,J];
                    X:=XIN[TEMP];
                    Y:=YIN[TEMP];
                    Z:=ZIN[TEMP];
                    DO_CURRENT_TRANSFORMATION(X,Y,Z);
                    AX[J]:=VPTX;
                    AY[J]:=VPTY;
                    AZ[J]:=VPTZ;
                END;
                POLYGON(AX,AY,AZ,SIDES);
            END;
        END;
        SETUP_VIEWPLANE(2.5,2.5,2.5,1,2,3,32,0,1,0);
        SET_PROJECTION_PARAMETERS(0,0,1,FALSE);
        SET_DEPTH_FLAGS(TRUE,TRUE);
        SET_WINDOW(-5,5,-5,5);
        SET_VIEW_DEPTH(0,50);
        SET_VIEWPORT(0,780,0,780);
        TOTAL_POINTS:=8;
        TOTAL_FACES:=6;
        SIDES:=4;
        INSPECT NEW INFILE("HOLLOW.DAT") DO
        BEGIN
            BUF:-BLANKS(20);
            OPEN(BUF);
            INIMAGE;
            FOR I:=1 STEP 1 UNTIL TOTAL_POINTS DO
            BEGIN
                INIMAGE;
                XIN[I]:=ININT;
                YIN[I]:=ININT;
                ZIN[I]:=ININT;
            END;
            FOR I:=1 STEP 1 UNTIL TOTAL_FACES DO
            BEGIN
                INIMAGE;
                FOR J:=1 STEP 1 UNTIL SIDES DO P[I,J]:=ININT;
            END;
            CLOSE;
        END;
        LOOP2:BLOCK;
        TRANSLATE_I(4,0,0);
        BEGIN_XFORM;BLOCK;END_XFORM;
        TRANSLATE_I(0,4,0);
        BEGIN_XFORM;BLOCK;END_XFORM;
        TRANSLATE_I(0,0,4);
        BEGIN_XFORM;BLOCK;END_XFORM;
        TRANSLATE_I(4,0,4);
        BEGIN_XFORM;BLOCK;END_XFORM;
        TRANSLATE_I(0,4,4);
        BEGIN_XFORM;BLOCK;END_XFORM;
        TRANSLATE_I(4,4,0);
        BEGIN_XFORM;BLOCK;END_XFORM;
        TRANSLATE_I(4,4,4);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(3,1,1);
        TRANSLATE_I(1,0,0);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(3,1,1);
        TRANSLATE_I(1,0,4);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(3,1,1);
        TRANSLATE_I(1,4,4);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(3,1,1);
        TRANSLATE_I(1,4,0);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(1,3,1);
        TRANSLATE_I(0,1,0);
        BEGIN_XFORM;BLOCK;END_XFORM;
```

```
        SCALE_I(1,3,1);
        TRANSLATE_I(0,1,4);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(1,3,1);
        TRANSLATE_I(4,1,4);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(1,3,1);
        TRANSLATE_I(4,1,0);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(1,1,3);
        TRANSLATE_I(0,0,1);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(1,1,3);
        TRANSLATE_I(4,0,1);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(1,1,3);
        TRANSLATE_I(4,4,1);
        BEGIN_XFORM;BLOCK;END_XFORM;
        SCALE_I(1,1,3);
        TRANSLATE_I(0,4,1);
        BEGIN_XFORM;BLOCK;END_XFORM;
        ELIMINATE_HIDING;
        POST_SEGMENT(0);
        UPDATE_DISPLAY;
        MOVE_CURSOR(800,750);
        OUTTEXT("PRESS <CR> TO ");
        OUTIMAGE;
        MOVE_CURSOR(800,720);
        OUTTEXT("CONTINUE");
        OUTIMAGE;
        INIMAGE;
        INIT_SEGMENTS;
        OUTTEXT("DO YOU WISH TO VIEW THE HOLLOW CUBE ");
        OUTTEXT(" FROM A DIFFERENT LOCATION");
        OUTIMAGE;
        OUTTEXT("TYPE 'YES' OR 'NO' IN TTY");
        OUTIMAGE;
        LOOP1:INIMAGE;
        CH:=INCHAR;
        IF CH='Y' THEN
        BEGIN
            OUTTEXT("SPECIFY 'VIEW-PLANE NORMAL' IN INTEGER UNITS");
            OUTIMAGE;
            INIMAGE;
            NX:=ININT;
            NY:= ININT;
            NZ:=ININT;
            SETUP_VIEWPLANE(2.5,2.5,2.5,NX,NY,NZ,32,0,1,0);
            CLEAR_SCREEN;
            DRAW_VIEWPORT;
            GOTO LOOP2;
        END
        ELSE
        BEGIN
            IF CH='N' THEN GOTO LAST
            ELSE
            BEGIN
                OUTMESSAGE;

                GOTO LOOP1;
            END;
        END;
        LAST:
    END;
END
```

```
FILE:HOLLOW.DAT
0 0 0
1 0 0
0 1 0
0 0 1
1 0 1
0 1 1
1 1 0
1 1 1
 8 6 4 5
 3 7 2 1
 7 8 5 2
 6 3 1 4
 7 3 6 8
 2 5 4 1
```

```
!FILE NAME IS DEMO.SIM;
BEGIN
    EXTERNAL PROCEDURE RUBOUT;
    EXTERNAL CLASS GRAS;..
    GRAS BEGIN
        TEXT BUF;
        INTEGER I;
        REAL TEMP;
        REAL ARRAY
        TX,TY,TZ,BX,BY,BZ,LX,LY,LZ,RTX,RTY,RTZ,FX,FY,FZ,RRX,RRY,
        RRZ[1:4];
        PROCEDURE CUBE;
        BEGIN
            A_POLYGON_3(FX,FY,FZ,4);
            A_POLYGON_3(LX,LY,LZ,4);
            A_POLYGON_3(RTX,RTY,RTZ,4);
            A_POLYGON_3(TX,TY,TZ,4);
            A_POLYGON_3(BX,BY,BZ,4);
            A_POLYGON_3(RRX,RRY,RRZ,4);
        END;
        INSPECT NEW INFILE("DEMO.DAT") DO
        BEGIN
            BUF:-BLANKS(15);
            OPEN(BUF);
            FOR I:=1 STEP 1 UNTIL 4 DO
            BEGIN
                INIMAGE;
                TX[I]:=ININT;
                TY[I]:=ININT;
                TZ[I]:=ININT;
            END;
            FOR I:=1 STEP 1 UNTIL 4 DO
            BEGIN
                INIMAGE;
                BX[I]:=ININT;
                BY[I]:=ININT;
                BZ[I]:=ININT;
            END;
            FOR I:=1 STEP 1 UNTIL 4 DO
            BEGIN
                INIMAGE;
                LX[I]:=ININT;
                LY[I]:=ININT;
                LZ[I]:=ININT;
            END;
            FOR I:=1 STEP 1 UNTIL 4 DO
            BEGIN
                INIMAGE;
                RTX[I]:=ININT;
                RTY[I]:=ININT;
                RTZ[I]:=ININT;
            END;
            FOR I:=1 STEP 1 UNTIL 4 DO
            BEGIN
                INIMAGE;
                FX[I]:=ININT;
                FY[I]:=ININT;
                FZ[I]:=ININT;
            END;
            FOR I:=1 STEP 1 UNTIL 4 DO
            BEGIN
                INIMAGE;
                RRX[I]:=ININT;
                RRY[I]:=ININT;
                RRZ[I]:=ININT;
            END;
            CLOSE;
        END;
        SETUP_VIEWPLANE(0.5,0.5,0.5,0,0,1,1,0,1,0);
        SET_PROJECTION_PARAMETERS(2,2,2,TRUE);
        SET_DEPTH_FLAGS(TRUE,TRUE);
        SET_VIEW_DEPTH(0,3);
        SET_WINDOW(-0.25,1.25,-0.25,1.25);
        SET_VIEWPORT(0,390,390,780);
        CUBE;
        SETUP_VIEWPLANE(0.5,0.5,0.5,1 ,0,1,1.5,0,1,0);
        SET_PROJECTION_PARAMETERS(0,2,1,TRUE);
        SET_WINDOW(-0.5,0.5,0.6,1.6);
        SET_VIEWPORT(390,780,390,780);
        CUBE;
        SETUP_VIEWPLANE(0.5,0.5,0.5,1,1,1,1.5,0,1,0);
        SET_PROJECTION_PARAMETERS(0,0,1,TRUE);
        SET_WINDOW(-0.5,0.5,-0.5,0.5);
        SET_VIEWPORT(0,390,0,390);
        CUBE;
        POST_SEGMENT(0);
        UPDATE_DISPLAY;
        SET_VIEWPORT(390,780,0,390);
        MOVE_CURSOR(400,360);
```

```
        OUTTEXT("TOP LEFT:");
        OUTIMAGE;
        MOVE_CURSOR(460,340);
        OUTTEXT("ONE POINT PERSPECTIVE");
        OUTIMAGE;
        MOVE_CURSOR(400,260);
        OUTTEXT("TOP RIGHT:");
        OUTIMAGE;
        MOVE_CURSOR(460,240);
        OUTTEXT("TWO POINT PERSPECTIVE");
        OUTIMAGE;
        MOVE_CURSOR(400,160);
        OUTTEXT("BOTTOM LEFT:");
        OUTIMAGE;
        MOVE_CURSOR(460,140);
        OUTTEXT("THREE POINT PERSPECTIVE");
        OUTIMAGE;
        MOVE_CURSOR(800,390);
        OUTTEXT("PRESS <CR> TO");
        OUTIMAGE;
        MOVE_CURSOR(800,370);
        OUTTEXT("CONTINUE");
        OUTIMAGE;
        INIMAGE;
        INIT_SEGMENTS;
        SETUP_VIEWPLANE(0.5,0.5,0.5,1,1,1,1.5,0,1,0);
        SET_PROJECTION_PARAMETERS(0,0,1,FALSE);
        SET_WINDOW(-1,1,-1,1);
        SET_VIEWPORT(0,390,390,780);
        CUBE;
        TEMP:=SQRT(0.5);
        SETUP_VIEWPLANE(0.5,0.5,0.5,0,0,1,1.5,0,1,0);
        SET_PROJECTION_PARAMETERS(TEMP,TEMP,1,FALSE);
        SET_WINDOW(0,2,0,2);
        SET_VIEWPORT(390,780,390,780);
        CUBE;
        TEMP:=TEMP/2;
        SET_PROJECTION_PARAMETERS(TEMP,TEMP,1,FALSE);
        SET_WINDOW(-0.5,1.5,-0.5,1.5);
        SET_VIEWPORT(0,390,0,390);
        CUBE;
        POST_SEGMENT(0);
        UPDATE_DISPLAY;
        SET_VIEWPORT(390,780,0,390);
        MOVE_CURSOR(400,360);
        OUTTEXT("TOP LEFT:");
        OUTIMAGE;
        MOVE_CURSOR(460,340);
        OUTTEXT("ISOMETRIC PROJECTION");
        OUTIMAGE;
        MOVE_CURSOR(400,260);
        OUTTEXT("TOP RIGHT:");
        OUTIMAGE;
        MOVE_CURSOR(460,240);
        OUTTEXT("CABINET PROJECTION");
        OUTIMAGE;
        MOVE_CURSOR(400,160);
        OUTTEXT("BOTTOM LEFT:");
        OUTIMAGE;
        MOVE_CURSOR(460,140);
        OUTTEXT("CAVALIER PROJECTION");
        OUTIMAGE;
    END;
    INIMAGE;
END
```

```
FILE:D[illegible].DAT
1 1 0
0 1 0
0 1 1
1 1 1
1 0 0
1 0 1
0 0 1
0 0 0
0 1 1
0 1 0
0 0 0
0 0 1
1 1 0
1 1 1
1 0 1
1 0 0
1 1 1
0 1 1
0 0 1
1 0 1
0 1 0
1 1 0
1 0 0
0 0 0
```